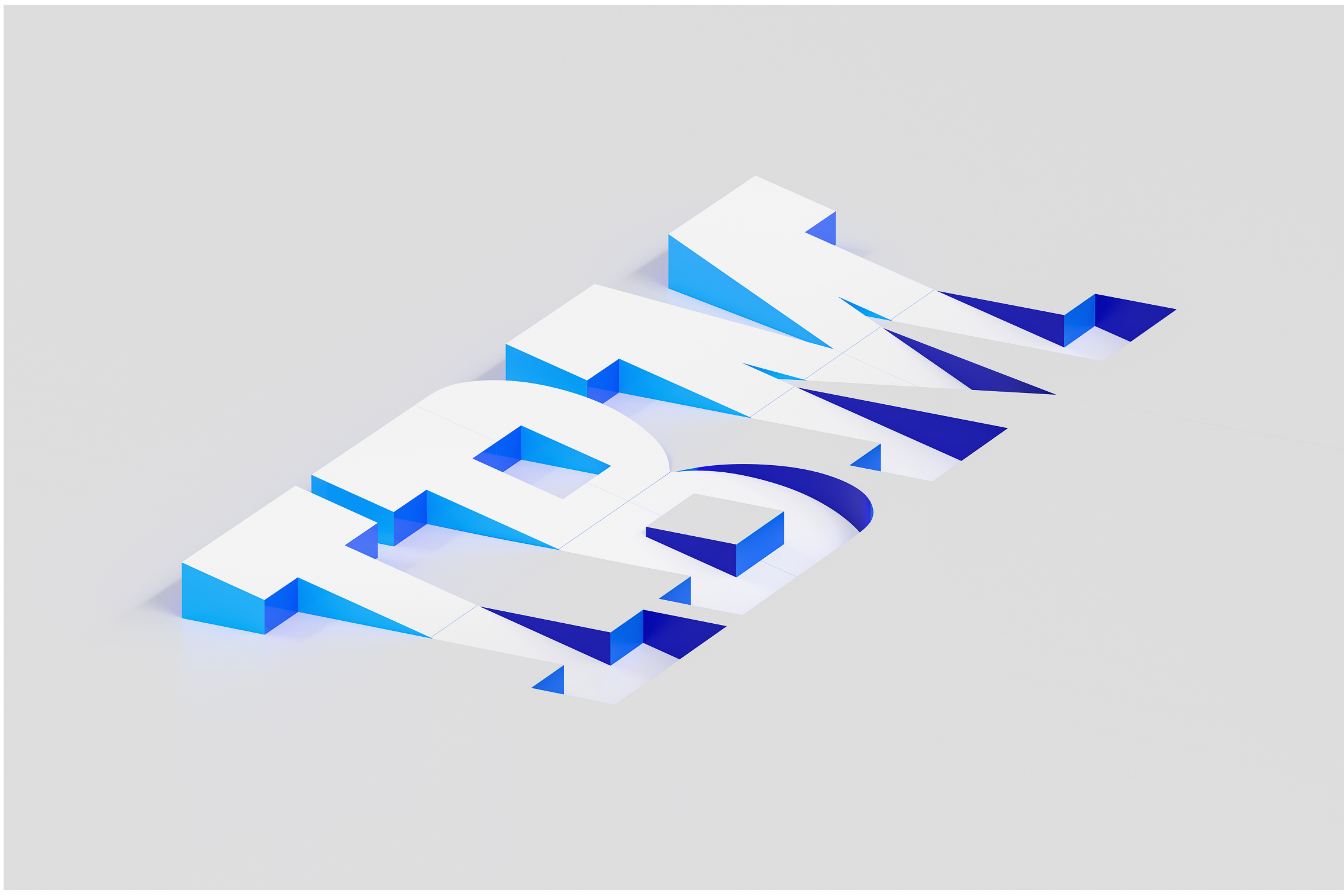
IBM Cloud

watsonx Assistant

Product guide



IBM

Edition notices

This PDF was created on 2025-10-21 as a supplement to *watsonx Assistant* in the IBM Cloud docs. It might not be a complete set of information or the latest version. For the latest information, see the IBM Cloud documentation at https://cloud.ibm.com/docs/watson-assistant.

Welcome to watsonx Assistant

Welcome to the documentation for watsonx Assistant!

IBM® watsonx™ Assistant, focused on using **actions** to build customer conversations, is designed to make it simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant can all now be done in one simple and intuitive interface.

- New **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- Each assistant has a **home page** with a task list to help you get started.
- Build conversations with **actions**, which represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer.
- A new way to **publish** lets you review and debug your work in a draft environment before going live to your customers.
- Use a new suite of **analytics** to improve your assistant. Review which actions are being completed to see what your customers want help with, determine if your assistant understands and addresses customer needs, and decide how can you make your assistant better.
- Explore our interactive demo site to learn how watsonx Assistant can be used to build powerful, scalable experiences for your users.

For more information, see FAQs about watsonx Assistant.

Visit Getting started with watsonx Assistant for a tutorial series on building in watsonx Assistant.

Switching between watsonx Assistant and the classic experience

You can easily switch back and forth between watsonx Assistant and the classic experience. However, watsonx Assistant provides a simplified user interface, an improved deployment process, and access to the latest features. For more information, see Migrating to watsonx Assistant. If you do need to switch, see Switching between watsonx Assistant and the classic experience.

Using watsonx Assistant

IBM® watsonx™ Assistant can be deployed as a managed cloud service or can be installed on premises. This documentation applies to a managed service on IBM Cloud or installed on IBM Cloud Pak for Data, and describes how to use the product regardless of how it is deployed. Information that applies exclusively to one deployment type is denoted by the appropriate tag:

- IBM Cloud for managed instances
- IBM Cloud Pak for Data version 4.6 or later for installed instances

About watsonx Assistant

Note: If you're looking for the AI Assistant Builder documentation, see Building AI Assistants.

Use IBM® watsonx™ Assistant to build your own branded live chatbot into any device, application, or channel. Your chatbot, which is also known as an *assistant*, connects to the customer engagement resources you already use to deliver an engaging, unified problem-solving experience to your customers.

	Benefit
Create AI-driven conversational flows	Your assistant uses industry-leading AI capabilities to understand questions that your customers ask in natural language. It uses machine learning models that are custom-built from your data to deliver accurate answers in real time.
Embed existing help content	You already know the answers to customer questions? Put your subject matter expertise to work. Add a search integration with IBM Watson® Discovery to give your assistant access to corporate data collections that it can mine for answers.
Connect to your customer service teams	If customers need more help or want to discuss a topic that requires a personal touch, connect them to human agents from your existing service desk provider.
Bring the assistant to your customers, where they are	Configure one or more built-in integrations to quickly publish your assistant on popular social media platforms such as Slack, Facebook Messenger, Intercom, or WhatsApp. Turn the assistant into a member of your customer support call center team, where it can answer the phone and address simple requests so live agents can address specific customer needs. Make your assistant the goto help resource for customers by adding it as a chat widget to your company website. If none of the built-in integrations fit your needs, use the APIs to build your own custom app.
Track customer engagement and satisfaction	Use built-in metrics to analyze logs from conversations between customers and your assistant to gauge how well it's doing and identify areas for improvement.

How it works

This diagram illustrates how IBM® watsonx™ Assistant delivers an exceptional, omnichannel customer experience:

How it works

Customers interact with the assistant through one or more of these channels:

- A web chat that you embed in your company website and that can transfer complex requests to a customer support representative
- An existing social-media messaging platform, such as Slack, Facebook Messenger, or WhatsApp
- A phone call or text message
- A custom application that you develop, such as a mobile app or a robot with a voice interface

The **assistant** receives a message from a customer and sends it down the appropriate resolution path.

If you want to process incoming messages, you can use webhooks to inject logic that calls an external service that can process the messages before the assistant routes them. Likewise, you can process responses from the assistant before they are returned to the customer.

The assistant chooses the appropriate resolution from among these options:

- An **action** interprets the customer's message further, then directs the flow of the conversation. The action gathers any information that it needs to respond or perform a transaction on the customer's behalf.
- A **search integration** uses existing FAQ or other curated content that you own to find relevant answers to customer questions.
- If a customer wants more personalized help or wants to discuss a sensitive subject, the assistant can connect the customer with someone from your support team through the web chat or phone integration.

To see how watsonx Assistant is helping enterprises cut costs and improve customer satisfaction today, go to Independent study finds IBM watsonx Assistant customers can accrue $23.9 million in benefits on the ibm.com blog.

Read more about these implementation steps by following these links:

- Building your assistant
- Publishing and deploying your assistant

Browser support

The watsonx Assistant application requires the same level of browser software as is required by IBM Cloud. For more information, see IBM Cloud Prerequisites.

For information about the web browsers that are supported by the web chat integration, see Browser support.

Language support

Language support by feature is detailed in Supported languages.

Getting started with watsonx Assistant

IBM® watsonx™ Assistant, which focuses on **actions** to build customer conversations, is simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant are all done in one simple and intuitive interface.

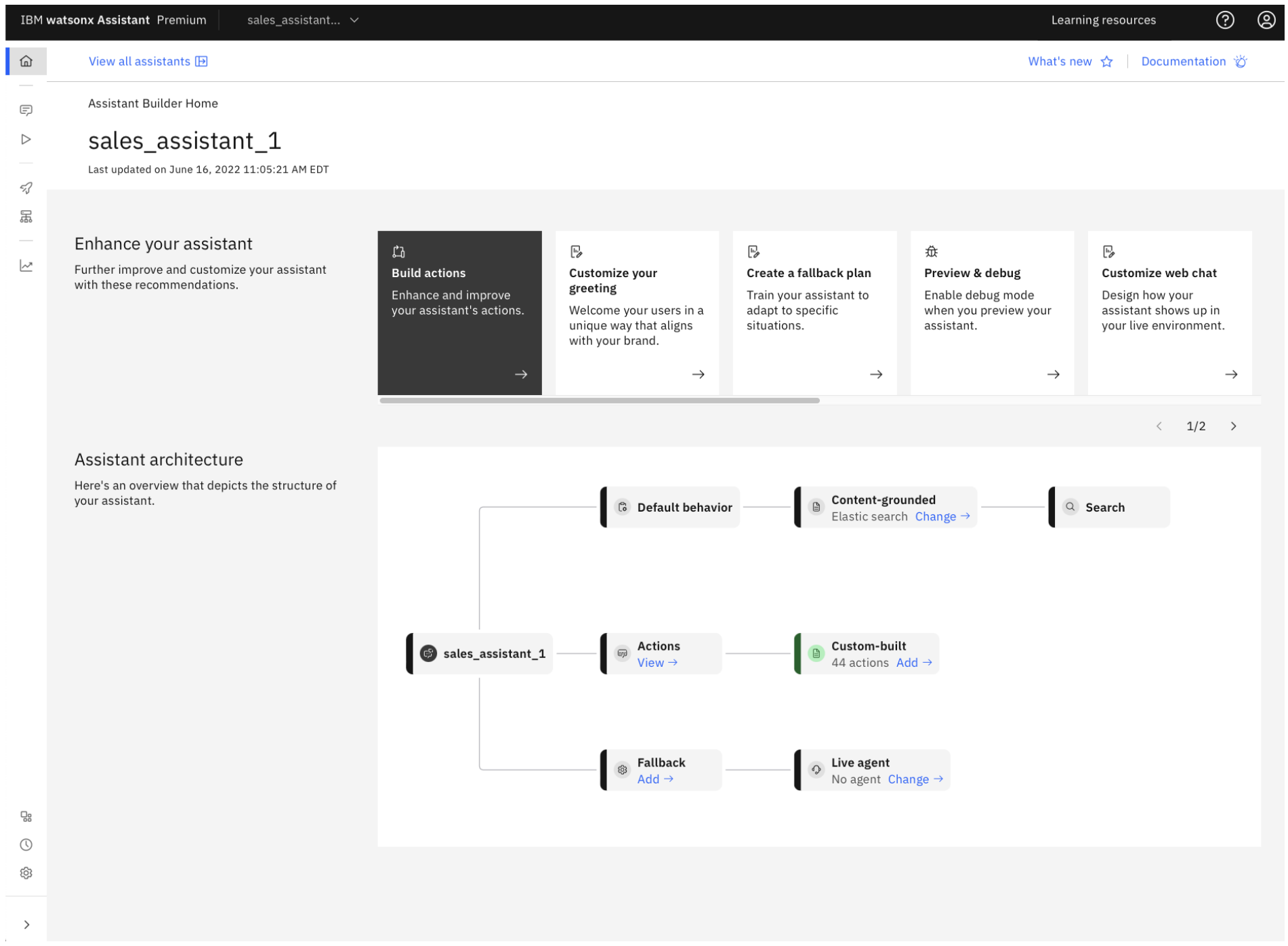
- The new assistant **architecture** enables simple navigation to specific parts of your assistant.
- The **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- The assistant has its own **home page** with a task list to help you get started.
- Build conversations with **actions** that represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer.
- **Publish** reviews and debugs your work in a draft environment before your assistant goes live to your customers.
- Use **analytics** to improve your assistant. Review which actions are being completed, determine whether your assistant understands and addresses customer needs, and identify ways to enhance your assistant's performance.

Home page

The **Home page** of watsonx Assistant provides you a complete overview of your assistant, settings, and configuration. In addition, you can quickly navigate to the following useful links from the **Home page**:

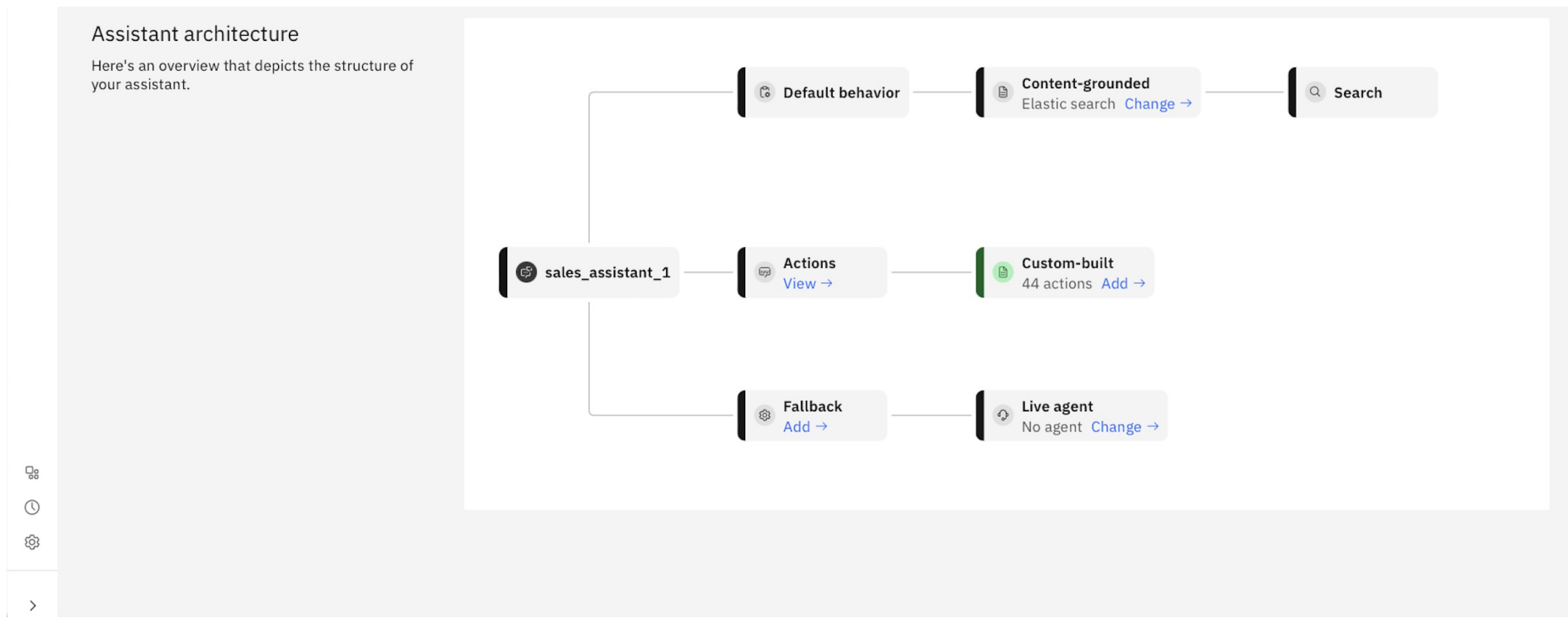
- Build actions
- Customize your greeting
- Set up search
- Preview & debug
- Customize web chat
- Set up a channel
- Set up live agent
- Publish your AI assistant
- Check analytics
- Create a fallback plan

enchanced-homepage



In the **Assistant architecture** section, you can view the complete structure of your assistant with links to actions that are created by you. In addition, you get a quick summary of your configuration that includes the number of actions that are created, the configured search integration, the integrated live agent, and the fallback configuration.

assistant-architecture



Tutorial

The following series of blog articles provide a tutorial to help you get started:

- Plan it
- Part I: The build guide
- Part II: Refine your assistant
- Part III: Test and deploy
- Part IV: preview, draft, publish, live

Some more starting points in the documentation to also help you get started:

- About watsonx Assistant
- Planning your assistant
- Overview: Editing actions
- Building actions from a template

For more information on FAQs, see FAQs about watsonx Assistant.

Working with your assistant

Assistants are created within a watsonx Assistant service instance. To continue working with your assistant, open the service instance that contains the assistant. If you can't remember the service instance name, you can switch between instances from within the watsonx Assistant user interface by doing the following steps:

1. Go to the IBM Cloud Resource list
2. Log in.

A list of the service instances that you own or were given access to is displayed.

3. Click a service instance to open it.
4. Click **Launch Watson Assistant** from the service instance details page to open the product in a new browser tab or window.
5. Click the following icon from the navigation pane to see a list of your assistants.

- Assistants

If you do not see the assistant you are looking for, you can look for it in a different service instance.

Switching the service instance

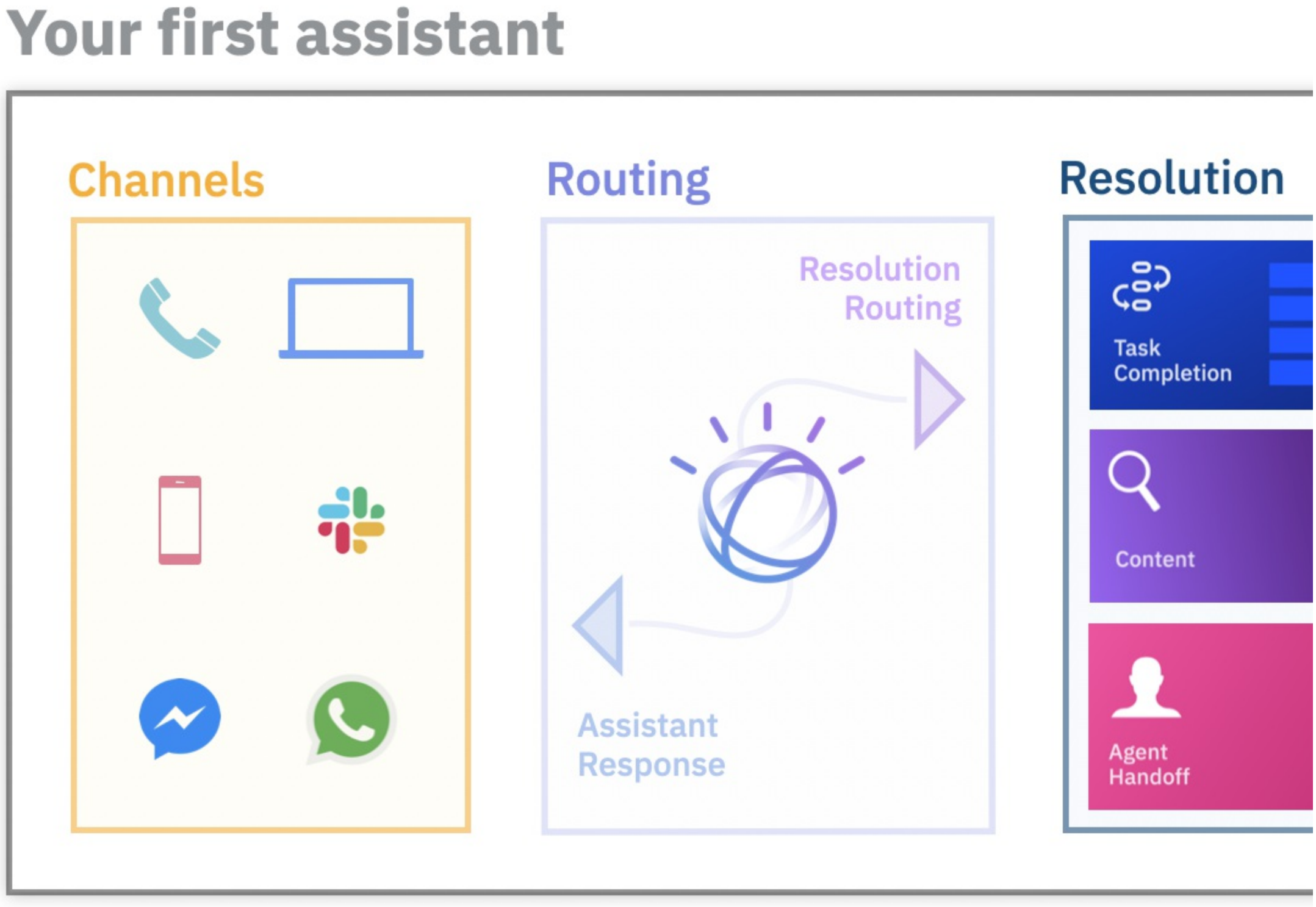
To switch to a different Watson Assistant service instance, complete the following steps:

1. Go to **Home**> **View all assistants**
2. You can see the list of the available service instances. Click on a different service instance of your choice to open.

Planning your assistant

Before you start to build an assistant, it’s important to think about your overall goals and plan how you're going to start.

Your assistant can use different resolution methods to help your customers with their requests. Customers access the assistant through channels that you choose and configure. The following diagram shows the structure of an assistant after you build it:



Consider these planning steps and make key decisions up front to keep you on track as you build.

1. Select an initial channel

Before you decide which specific topics to build into your assistant, you must first decide where to deploy your assistant so that customers can easily find it. For example, you can embed the assistant in your company website or add it to a messaging platform such as Facebook, Slack, or WhatsApp. The primary channels that you can use to communicate with your customers are your website and the phone.

If you choose to use web chat to communicate with customers on your website, decide on which of your web pages you want the assistant to appear. To start, you might identify the pages where your customers most frequently ask questions from your customer service team.

If you have an existing interactive voice response (IVR) system with a branching structure (“press 1 for billing, press 2 for payments”), you might choose the phone integration as your initial channel. You can integrate watsonx Assistant with your existing system to automate the IVR experience so customers can talk with your assistant over the phone.

Understanding where you will deploy the assistant before you begin can help you author the right types of answers for a given channel or platform. For more information about ways to deploy an assistant, see Adding integrations.

2. Pick your assistant's domain of expertise

Decide which general domain of expertise you want your assistant to cover (for example, billing support or scheduling appointments). To make an informed decision, review any support call logs that you have access to or ask your customer service representatives. After you choose a domain, be sure

that it aligns with a channel that you can control and change. For example, don’t choose to automate billing support questions if you're unable to add the web chat client to the billing web pages.

After you select a domain, you can decide which specific questions or tasks the assistant will help customers with. Start small. Pick one or a handful of customer issues that will deliver the highest value to start. It might be valuable for your assistant to answer a simple question that is asked all the time. Or maybe there's a task, such as scheduling appointments, that you can offload to the assistant to tackle incoming customer requests.

Choose a narrow set of user goals first. If you start small and choose the goals with the highest impact first, you'll have room and time to grow the expertise of your assistant. After your assistant is live, the built-in metrics of active user conversations help you understand what your customers are asking about, how well your assistant is able to meet their needs, and what to focus on next.

3. Choose the tone and language of your assistant

Before you build your assistant, it can also be helpful to choose the tone with which your assistant communicates. Is your assistant an optimist or a pessimist, a shy intellectual type, or an upbeat sidekick? Write conversations that reflect your assistant's personality. Don't overdo it by sacrificing usability for the sake of keeping your assistant in character. Strive to present a consistent tone and attitude.

Never misrepresent the assistant as being a human. If users believe that the assistant is a person, then find out it's not, they are likely to distrust it. In fact, some US states have laws that require chatbots to identify themselves as chatbots.

Decide whether and how you want to handle more than one spoken language. For more information about ways to approach language support, see Adding support for global audiences.

4. Connect to content sources

The answer to common questions might already be documented somewhere in your organization's technical information. Plan whether you want to connect the assistant to existing content sources by taking an inventory of relevant help content (for example, product information, knowledge articles, FAQs) available to your customers.

You can give your assistant access to this information by adding a search integration to your assistant. The search integration uses IBM Watson® Discovery to return smart answers to natural language questions.

5. Plan your handoff strategy

Finally, prevent customers from hitting dead ends by escalating conversations to a human agent if the assistant isn't able to resolve a customer's question or problem. As part of your planning, figure out what your escalation strategy is going to be. This strategy will vary depending on the deployment channel you choose.

If you deploy to your website, you have three options for escalating to a human agent:

- Inline
- Email address
- Phone number

If you opt for the inline approach, you can directly escalate to a human agent in your existing contact center tool without forcing the user to leave the web chat widget. This approach also provides the agent with the full context of the conversation. To use either the email address or phone number approach, provide the user with the agent’s email address or phone number. These approaches are simple to set up, but they can be more disconnected experiences for your customers.

If you deploy by using the phone integration, you can reach a human agent only by transferring the phone call to someone who can help.

Start building an assistant

If you're ready to start building an assistant, see Editing actions for more information.

Release notes

Release notes for watsonx Assistant

Release notes describe the new features, changes, and bug fixes in each release of the product. For more information about changes in the web chat integration, see the Web chat release notes.

05 Aug 2025

Withdrawal of IntelePeer Phone Number Service

The IntelePeer free phone number service for watsonx Assistant is withdrawn from 31 July 2025 (refer to the earlier announcement). Integrating your assistant with those phone numbers no longer works. Migrate to Twilio or another provider to avoid disruption. For assistance, contact support at https://www.ibm.com/mysupport/. For more information see, Integrating with phone and Integrating with *SMS*.

04 June 2025

Feature parity for watsonx Assistant

Know about the feature parity between the software as a service (SaaS) and the on-premises offerings for the available features in watsonx Assistant. For more information, see Feature parity.

02 June 2025

Autolearning feature deprecation

The autolearning feature in watsonx Assistant is discontinued from June 16, 2025. The settings are removed, and all autolearning capabilities are disabled. For more information see, Using autolearning to improve assistant responses.

22 May 2025

IntelePeer phone number service deprecation

The IntelePeer free phone number service for watsonx Assistant is discontinued from 31 July 2025. Migrate to Twilio or another provider to avoid disruption. For assistance, contact support at https://www.ibm.com/mysupport/. For more information see, Integrating with phone and Integrating with *SMS*.

15 May 2025

Documentation update: Preventing data use for Watson services

You can prevent Watson services from using your data for service improvements by adjusting your privacy settings. For more information, see Controlling request logging for Watson services.

10 April 2025

Display name now shown for disambiguation

You can now provide a friendly name as display name for you actions. Users will see the display name during actions disambiguation, which is when AI matches multiple actions and provides a list of options for users to select. For more information, see Overview: Editing actions.

New Signed JWT authentication method

The Signed JWT authentication is no longer the default when you create a webhook. Instead, you can select it in the **Edit authentication** dialog. For more information, see Defining the authentication method for pre-message and post-message webhooks .

27 March 2025

Enhance security with audio recording in phone integration

You can now use audio recording with the Genesys Audio Connector in Phone Integration! To learn more, see Recording a caller's utterance.

13 March 2025

Webhook authentication configuration

watsonx Assistant now supports Webhook authentication to verify that the Pre-message webhooks and Post-message webhooks requests are legitimate, comes from the expected source, and ensures the authenticity and integrity of the data. For more information, see Defining the authentication method for pre-message and post-message webhooks.

Update on the search confidence for conversational search

Small modifications to text processing can impact the values of retrieval confidence, response confidence, and extractiveness. For more information on the scores, see Conversational search analytics.

27 February 2025

Conversational search language support

Conversational search now supports four languages other than English, including French, Spanish, German, and Brazilian Portuguese. For more information, see Languages supported for conversational search.

18 February 2025

Testing limits update

There are no longer limits to the amount of tests, but there is still a limit of 250 messages per test. For more information, see Testing limits for the set of data.

New Generative AI page

You now have a new Generative AI page that helps you to configure large language models(LLMs) for your assistants. The LLMs enable your customers to interact with the assistants seamlessly without any custom-built conversational steps. For more information, see Generative AI.

21 January 2025

Setting translation entry limit in multilingual package

To successfully download the multilingual package, the total number of translation entries in your assistant must be less than or equal to 400000. For more information, see Using multilingual downloads for translation.

17 January 2025

Updated conversational search user query

Removed a question mark (`?`) that was automatically added to the user query in conversational search. This fixes some edge cases for using elasticsearch integration. It might rarely alter the conversational search responses.

13 December 2024

Direct search integration support for Milvus

You can now directly configure your search integration with Milvus. Milvus is a high-performance, highly scalable vector database that you can use for Conversational Search. For more information, see Milvus search integration setup.

02 December 2024

New version 2024-12-02 for API

Microsoft Teams integration now supports `options` and `suggestions` response types to have distinct `label` (display name) and `value` (response sent back to watsonx Assistant upon selection). Previously, the same string was used for both `label` and `value`.

26 November 2024

Conversational search watsonx.ai model deprecation

The model `granite-13b-chat-v2` was deprecated on 04 November 2024 and will not be available after 19 January 2025. On 19 January 2025, `granite-3-8b-instruct` becomes the default model, automatically replacing the deprecated model.

22 November 2024

Post process conversational search responses

You can now save your conversational search responses in action variables for post-processing. For more information, see Search for the answer.

Evaluate assistant

You can now evaluate the performance of your assistant. For more information, see Evaluating the assistant.

6 November 2024

Skill output display

You can now choose to hide or display the skill output that is stored as an assistant variable. For more information, see Passing values to a subaction.

21 October 2024

New table response type

watsonx Assistant now supports the table response type. This is a beta version that is currently only supported in the watsonx Assistant web chat. For more information, see Response types reference.

11 October 2024

Passing values to skill-based actions

watsonx Assistant now supports passing values to skill-based actions in the conversation flow. External skill providers must implement the new endpoint for skill-based actions to use this feature. For more information, see Passing values to a subaction.

Inspector supports conversational search

You can now debug conversational search issues using the **Inspector**. For more information, see Debugging failures for Conversational search or skill based actions.

8 October 2024

Improved currency recognition

watsonx Assistant now supports filling currency slots with number entities in conversational skills.

4 October 2024

Configure Milvus in Custom service integration

You can now configure Milvus, a full-text search engine with high performance and real-time analytics capabilities, in the Custom service integration of your assistant for enhanced search. For more information, see Setting up Milvus for Custom service.

13 September 2024

Conversational search and slot filling in Enterprise

Intelligent Information gathering and Conversational search are now available for **Enterprise with data isolation plan**.

6 September 2024

Access properties from stream events

You can now access properties from the last event in a stream more efficiently. For more information, see Running the stream.

Conversational search debugging

You can now troubleshoot the errors when you face problems in conversational search. For more information, see Debugging failures

31 August 2024

Conversational search analytics

You can now analyze the performance of your conversational search by using conversational search analytics. For more information, see Conversational search analytics.

30 August 2024

Enhancement in analytics conversations

Your assistant is now enhanced to provide history of conversations by specifically indicating the different types of actions, search, dialog, Conversational search, and intents. For more information, see Review customer conversations.

28 August 2024

Contextual awareness in conversational search

You can now choose the type of conversational search based on the context by using contextual awareness. For more information, see Contextual awareness.

27 August 2024

Security certificates configuration

You can now secure the search integrations, custom extensions, conversational skills, and webhooks by configuring TLS certificates in your assistant. For more information, see Configuring Security certificates.

Source chunk visibility in citations

You can now see full source chunk in citations even when PDF viewer and source link to a website is not available.

25 August 2024

New version 2024-08-25 for the search APIs

The search results are no longer returned in the `results` property. Instead, two new properties are introduced:

- `primary_results` - Returns the initial search results from a query. The result limit is configured using `max_primary_results`, which sets how many search results are included in the `primary_results` property.
- `additional_results` - Returns more or additional search results from a query. The result limit is configured using `max_total_results`, which sets how many search results are included in the `additional_results` property.

20 August 2024

Intelligent Information gathering

This release brings an improvement of the accuracy of filling free-text step variables for information gathering.

9 August 2024

Replace OpenAPI document for custom extension

You can now replace the OpenAPI document of a custom extension to change the extension behavior and authentication method. For more information see, the Building the custom extension and Replacing OpenAPI document topics.

31 July 2024

Information gathering is now generally available

The watsonx.ai information gathering capability in watsonx Assistant is now generally available. For more information, see Information gathering.

30 July 2024

New Custom service integration for search

You can now use Custom service integration to create your own search capability in your assistant and enhance the AI responses through conversational search capabilities. For more information, see Custom service integration.

New option to define the response length in conversational search

You can now define the length of generated responses. For more information, see Conversational search.

29 July 2024

New supported languages for information gathering

The watsonx.ai information gathering capability in watsonx Assistant now supports French, German, Spanish, Brazilian Portuguese, and Japanese. For more information, see Information Gathering and Global settings for actions.

23 July 2024

New "End the action" step behavior

Using "End the action" in the last step no longer ends the action immediately. For more information see, Choosing what to do at the end of a step .

New Hate, Abuse, and Profanity Filter for watsonx Assistant

The new Hate, Abuse, and Profanity (HAP) filter in watsonx Assistant enhances user safety by identifying and addressing hate speech, abuse, and profanity. This feature helps maintain a positive and inclusive online environment for your customers. For more information, see HAP filter.

18 July 2024

Server-Sent Events for custom extensions

You can now define a JSON Path to stream text from Server-Sent Events (SSE) in custom extensions. This feature ensures that text is streamed and stored correctly, with built-in checks to handle potential errors. For more information, see Streaming from an extension.

03 July 2024

Referencing user-defined variables with `user_defined_` prefix

You can now reference the user-defined action or session variables through expressions in actions by using a reserved keyword prefix `user_defined_`. For more information, see Variables.

01 July 2024

All-new homepage for watsonx Assistant

The all-new homepage of watsonx Assistant has well-organized sections and easy-to-access tabs to enhance your user-interface experience. For more information, see Getting started.

7 June 2024

Improved accuracy in information gathering

The information gathering feature is enhanced to reduce errors while filling the free-text step variables from the customer response. This allows for more accurate and efficient data entry. For more information, see Information gathering.

4 June 2024

Phone-based conversational search support

You can now use the streaming feature in the conversational search for the phone integration in your assistant. For more information see, Conversational search and Integrating with phone.

14 May 2024

Autocorrection is disabled for search integration

The autocorrection feature is now disabled for search integration in your assistant. For more information, see Autocorrecting user input.

03 May 2024

Conversational search is now generally available

The Conversational search feature with the Retrieval-Augmented Generation (RAG) solution, powered by watsonx.ai, is now generally available. For more information about conversational search, see Conversational search.

Elasticsearch search integration is now generally available

Elasticsearch search integration is now generally available. For more information about Elasticsearch search integration, see Elasticsearch search integration setup.

18 April 2024

Tuning your assistant's tendency to say "I don't know"

You can now tune the tendency of your assistant to say "I don't know" in conversational search by using the **Tendency to say “I don’t know”** option in the conversational search settings. This option can help to reduce Large Language Model (LLM) hallucinations and provide higher fidelity answers for conversational search by tuning your assistant's tendency to fall back to the “I don’t know” answer. For more information, see Tuning conversational search’s tendency to say “I don’t know".

15 April 2024

Streaming response for Conversational search

You can now use streaming response in your assistant for conversational search with the help of watsonx.ai capabilities to provide continuous and real-time responses. For more information, see Streaming response for conversational search.

Overwrite all or skip all when you copy actions to another assistant

You can now choose to overwrite all references or skip all references when you copy actions from one assistant into another. For more information, see Copying an action to another assistant.

11 April 2024

Add examples for variables in the action editor

You can now add examples of variables in a user input by using the Add examples button in the action editor. This helps your assistant to retrieve accurate information and respond to the user quickly. For more information, see Adding examples for clarity.

Add a custom result filter for the search integration

You can now filter your search result in the Discovery search integration by adding custom text strings in the **Custom result filter** field in **Search integration**. For more information, see Configure the search for Discovery.

Configure search routing

You can now configure the search routing for your assistant when no matches are available for the customer response. For more information, see Configuring the search routing when no action matches.

26 March 2024

Support integration with Genesys Audio Connector

You can now integrate Genesys Audio Connector with your assistant to stream the conversation audio between the assistant and Genesys Cloud. For more information, see Integrating with Genesys Audio Connector.

16 February 2024

Support for Llama2 chat format

You can now transform the session history of your assistant to a Llama2 compatible format for the generative AI use cases. For more information, see `Array.transform()` and Session history.

12 January 2024

Segment extension

The segment extension is now available for Plus plan, Enterprise plan and Enterprise with data isolation plan. For more information, see Sending events to Segment.

View number of actions within collections

You can now view the total number of actions that are within each collection on the **Action** page.

Search and analyze the conversational search requests

You can now search and analyze the conversational search requests from the customer in the **Analyze** page. For more information, see Filtering conversations.

15 December 2023

OpenAPI document file size limit for integrating custom extension

When you integrate a custom extension by using REST API, the maximum file size of the OpenAPI document that you can import is limited to `4 MB` if you have a *Plus* or *Enterprise* plan of watsonx Assistant. However, if you have an Enterprise plan with data isolation, the maximum file size of the document is limited to `8 MB`. For more information, see Preparing the API definition.

Edit variable values directly in debug mode

You can now edit the variable value directly in the debug mode by clicking on the values. For more information, Editing the variable values.

Expand the debug mode panel for better visibility

You can now expand the debug mode panel by clicking the **Expand** icon for better visibility of the long variable values. For more information, see Variable values in Preview.

12 December 2023

Invalid date entry not accepted

Starting from this release, the assistant does not recognize the invalid date entry such as `Feb 31`, `31/11/2022`, and `Feb 29 2023`. In addition, the assistant does not automatically parse the invalid dates to the first day of the month. For more information, see @sys-date.

8 December 2023

Algorithm version Latest(15-Apr-2023) uses improved intent detection and matching

The **Latest(15-Apr-2023)** algorithm version uses a new foundation model to improve the intent detection and action matching in assistants with languages such as English, French, German, Portuguese (Brazilian), Spanish, Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, Italian, Japanese, and Korean. The new foundation model is trained by using the transformer architecture. For more information, see Algorithm version and training.

Change in Pause response time configuration

You can now pause a response for any duration from `0` to `60 seconds`, which was limited from `1` to `10 seconds` in the previous releases. In addition, you can now pause the response in `milliseconds` by entering decimals of a `second` in the **Duration** field. For more information, see Pause response.

28 November 2023

New Genesys Bot Connector integration

You can create a Bot Connector that integrates Genesys Cloud with your assistant to faciliate messaging and conversation flows. For more information, see Integrating with Genesys Bot Connector.

Intelligently recognize information in free text responses

You can now enable your assistants to intelligently recognize information provided by the customer in a session when using the free text response in an **Action** step. When you enable this feature, your assistant uses a large language model (LLM) in watsonx.ai to intelligently recognize multiple pieces of information in customer responses and fill corresponding steps to avoid multiple prompts in a session. This is a beta feature and available only in assistants that use the English language. For more information, see Information Gathering and Global settings for actions.

Conversational search by integrating Elasticsearch

You can now configure Elasticsearch as the content source for your search integration. For more information, see Adding search.

20 November 2023

Message logs for a dialog without action

You can now use the **Message logs** tab in the **Analyze** page to see a log of all messages that are sent between the user and the assistant by turns. This feature is available for all messages including the messages that are sent when dialog is activated. For more information, see Review customer

conversations.

Masking confidential customer information in assistants with `stateless` endpoint

You can now mask the confidential information in user input and assistant responses for assistants with the `stateless` endpoint. In the `stateless` message API, the private data is encrypted before returning them to the client app. For more information, see Using variables to manage conversation information and Protecting the privacy of the customer information.

10 November 2023

Masking confidential customer information

You can now mask the confidential customer information in an action by marking the variables as private. The feature for masking the confidential customer information is available only for actions in assistants with the `stateful` endpoint. For more information, see Using variables to manage conversation information and Protecting the privacy of the customer information.

Editing and Visualization

After you create an action, you now have the choice to switch from the step edit view to a new visualization that displays a canvas with a flow chart of the action. Within the canvas you can pan and zoom. This visualization of an action is currently in beta. See Visualizing the flow of the action for more information.

18 October 2023

Unrecognized requests analysis for new languages

The unrecognized requests analysis is now available for two more languages, Spanish and Brazilian Portuguese. You can access this only for assistants on IBM Cloud if you are in the Plus or higher plan. For more information, see Use unrecognized requests to get action recommendations.

Improvements to dialog search

When you search intents, entities, and dialog nodes, accuracy is improved and sensitivity is adjusted. Now, the language matches the language of the created resources for highest accuracy. For example, searching for Japanese keywords should be done in a v1 workspace or v2 dialog with Japanese as the language set. A potential workaround is to allow partial matches. Also, searches are for exact matches, so the search engine is sensitive to underscores (_) and extra spaces in the keyword.

8 October 2023

Introducing watsonx Assistant

IBM Watson® Assistant is renamed. It is now called IBM® watsonx™ Assistant.

25 August 2023

Setting for when to use **No matches**

You can use a new global setting for actions to change how often your assistant routes customers to the **No matches** action. By setting this threshold, you can affect when the assistant fetches answers from a search integration, triggers the *Fallback* action, or switches topics. For more information, see When the assistant can't understand your customer's request.

24 August 2023

See who last edited a collection or action

Starting with this release, you can see who last edited a collection or action. On the Actions page, you can hover on the values in the Last edited

column to see the email address of the person who last modified the collection or action. This information is available for edits as of August 24, 2023.

Change to response mode default settings

As of this release, the default settings for *Step validation attempts before offering support* have changed. In *Clarifying* mode, the default is now 2 instead of 1. In *Confident* mode, the default is now 4 instead of 3. For Enterprise plans, the customization choices are now 2 times, 3 times, and 4 times instead of 1 time, 2 times, and 3 times. For more information, see Response modes.

Change to id values in multilingual downloads for translation

As of this release, there is a change to the `id` values in the multilingual downloads for translation. If you have used the multilingual download before, you need to download a new CSV file to match the IDs in your assistant. For more information, see Using multilingual downloads for translation.

10 August 2023

Algorithm version **Beta** provides improved intent detection and action matching for more languages

The algorithm version **Beta** now provides improved intent detection and action matching for Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, Italian, Japanese, and Korean. It includes a new foundation model that is trained using a transformer architecture to improve intent detection and action matching.

Improvements include:

- Improved robustness to variations in user inputs such as typos and different inflection forms
- Less training data required to reach the same level of performance compared to previous algorithms

For more information, see Algorithm version and training

Session history variable

You can record a summary of the recent messages from a conversation for each customer, for use with the `session_history` variable. Session history might be used to provide a summary of the conversation to a live agent during a transfer or call a generative AI extension to generate an answer based on a summary. For more information, see Session history.

3 August 2023

Synonym recommendations discontinued

Synonym recommendations for entities, in the classic experience for dialog skills, are discontinued as of this release.

27 July 2023

Trigger word detected action is now generally available

Use the **Trigger word detected** action to add words or phrases to two separate groups. The first group connects customers with an agent, when it's important for a customer to speak with a live agent rather than activate any further actions. The second group shows customers a customizable warning message, used to discourage customers from interacting with your assistant in unacceptable ways, such as using profanity. This action is included with all new assistants created as of February 2, 2023. This was a beta feature that is now generally available. For more information, see Detecting trigger words.

Action conditions are now generally available

An action condition is a Boolean test, based on some runtime value; the action runs only if the test evaluates as true. This test can be applied to any variable. By defining action conditions, you can do things such as control user access to actions or create date-specific actions. This was a beta feature that is now generally available. For more information, see Adding conditions to an action.

New expression method to sort arrays

The new `Array.sort()` method does an in-place sorting and returns the sorted array.

24 July 2023

OAuth 2.0 custom flow support for extensions

Any grant type that starts with `x-` is supported for OAuth 2.0 authentication. For more information, see Preparing the API definition and OAuth 2.0 Authentication.

20 July 2023

Action and collection names must now be unique

With this release, each action name must be different from other actions, and each collection name must be different from other collections. If your existing actions or collections have duplicate names, a warning icon appears in the Status column. For more information, see Overview: Editing actions and Organizing actions in collections.

14 July 2023

Multiple responses when customizing validation

When you edit validation for a customer response, you can now include several validation responses. For more information, see Customizing validation for a response.

Fallback choice for dynamic options

The *dynamic options* response type now includes a fallback static choice, such as `None of the above`, if the options aren't what the customer wants. You can then add a step that is conditioned on this static option to provide further assistance. For more information, see Dynamic options.

Allow change of topic between actions and dialog

If you are using actions and dialog, there is a new setting you can use to ensure that customers can change topics between an action and a dialog node. For more information, see Allow change of topic between actions and dialog.

Use multilingual downloads for translation

You can enable the download of language data files, in CSV format, so you can translate training examples and assistant responses into other languages and use in other assistants. For more information, see Using multilingual downloads for translation.

12 July 2023

Algorithm version **Beta** provides improved intent detection and action matching for more languages

The algorithm version **Beta** now provides improved intent detection and action matching for French, German, Portuguese (Brazilian), and Spanish. It includes a new foundation model that is trained using a transformer architecture to improve intent detection and action matching.

Improvements include:

- Improved robustness to variations in user inputs such as typos and different inflection forms
- Less training data required to reach the same level of performance compared to previous algorithms

For more information, see Algorithm version and training

10 July 2023

Change to message API

When debug is enabled on the runtime (`/message`) API call, if a dialog processing error happens, more output is presented within the `output.debug.log_messages` attribute.

7 July 2023

SpEL expression of type `intents[i].confidence` is deprecated in actions

On or after July 10, 2023, this expression will evaluate to -1 with a warning log message. On or after July 17, 2023, usage of this expression will be removed and will result in an error.

21 June 2023

Autolearning now available for all languages

Autolearning is now generally available for all languages and enabled by default in all new assistants. Your assistant can learn from interactions with your customers and improve responses. For more information, see Using autolearning to improve assistant responses.

See which actions use a specific variable

The Variables page now includes a new *Actions count* column. You can click on the number in the column to see which actions use a variable. For more information, see Creating a session variable.

Expression editor character limit increased

The character limit in the expression editor has increased from 1,024 to 4,000. For more information, see Writing expressions.

15 June 2023

New API version

The current API version is now 2023-06-15. This version introduces the following changes:

- Invalid expressions in callouts to client actions are now replaced with empty strings
- The default value of `async_callout` changes from `true` to `false`

Custom extensions now support OAuth 2.0 authentication

You can now use OAuth 2.0 authentication when you build and add custom extensions. For more information, see Building a custom extension and Adding an extension to your assistant.

8 June 2023

Edit step titles

You can now add and edit titles for each step, which can help you more easily identify what a step does in an action. For more information, see Editing actions.

New message when deleting a subaction

If you delete an action that is a subaction, there is a new message that lists all the actions that call the subaction and asks you to confirm the deletion. The message helps you preserve dependencies between actions and subactions.

5 June 2023

Filtering the list of actions

You can locate specific actions by filtering the list by subactions, by custom extension, or by variable. For more information, see Filtering actions.

29 May 2023

New expression choice for setting a session variable

Previously, to use an expression to set or modify a variable value, you needed to pick an existing variable or create a new one and select the expression option. Now you can use a new **Expression** choice to write an expression directly without first picking a variable. For more information, see Storing a value in a session variable .

Changes to validation of OpenAPI specifications

When you build a custom extension, you work with OpenAPI specification files. This release includes changes to the validation of OpenAPI files, which might affect the connection between your actions and extensions

You can import an action with references to a custom extension and watsonx Assistant can automatically connect the action and extension. With the validation change, modifications to the OpenAPI specification for your custom extension might affect this automatic connection. For more information, see Building a custom extension.

22 May 2023

Changes to the date and number formats in assistant responses

Beginning 22 May 2023, customers might see changes to the date and number formats in assistant responses.

Examples of date changes include removed or added periods, such as:

- In Spanish, `18 abr. 2021` changes to `18 abr 2021`
- In Portuguese, `18 de abr` changes to `18 de abr.`

The delimiter character changes for numbers in some languages. For example, in French, nonbreaking space (NBSP) changes to narrow no-break space (NNBSP).

These changes are the result of migrating the watsonx Assistant platform to Java 17, where locale values are updated by using specifications in CLDR 39.

To avoid or minimize the impact of similar changes in the future, you can use Display formats.

18 May 2023

Differences in contextual entity detection for dialog skills with few annotations

If you have 10 to 20 examples of contextual entities in your dialog skill, you might see differences in the entities detected due to updates made to address critical vulnerabilities. The impact of these differences is limited to only newly-trained models. Existing models are unaffected. You can mitigate these differences by annotating more examples. For more information, see Annotation-based method.

17 May 2023

Display iframe inline

In the web chat, there are now two ways an iframe response can be included:

- As a preview card that describes the embedded content. Customers can click this card to display the frame and interact with the content.
- Inline, meaning within the conversation. This new option is good for smaller pieces of iframe content.

For more information, see Adding an iframe response.

15 May 2023

Change to dialog skill context variables named `request`

If your dialog skill used a context variable that is named `request`, it was removed from the response payload of any `/message` calls in the V1 or V2 API, or through the watsonx Assistant user interface. After 15 May 2023, this behavior changes. Your assistant doesn't remove context variables that are named `request` from the response payload anymore.

5 May 2023

New validation choices for date, time, and numeric customer responses

For *Number*, *Date*, *Time*, *Currency*, and *Percentage* customer responses, you can now customize the validation to check for a specific answer, such as a range of dates or a limited currency amount. For more information, see Customizing validation for a response.

3 May 2023

Algorithm version **Beta** provides improved intent detection and action matching

The algorithm version **Beta** now provides improved intent detection and action matching. It includes a new foundation model that is trained using a transformer architecture to improve intent detection and action matching for English.

Improvements include:

- Improved robustness to variations in user inputs such as typos and different inflection forms
- Less training data required to reach the same level of performance compared to previous algorithms

For more information, see Algorithm version and training.

24 April 2023

Response modes randomization behavior

The response modes beta now uses the same randomization behavior during clarification that your actions have without response modes enabled. Previous to this change, when response modes were enabled, the clarification feature no longer periodically modified the options for clarification. Randomizing the clarification helps prevent bias that can be introduced by a percentage of people who always pick the first option without carefully reviewing all of their choices beforehand. For more information, see Response modes or Asking clarifying questions.

21 April 2023

Collections

You can use a *collection* to organize your actions. You can put actions into folder-style groups based on whatever categorization you need at your organization, such as by use case, internal team, or status. For more information, see Organizing actions in collections.

18 April 2023

Activity log

The **Activity log** is a beta feature that is available for evaluation and testing. Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant. It is available for Plus plans and higher. For more information, see Activity log.

16 April 2023

Allow changing topics in free text and regex responses

By default, customers can't change topics when the assistant is asking for a free text response or when an utterance matches the pattern in a regex response. Now free text and regex customer response types have a setting to allow a user to digress and change topics. For more information, see Enabling changing the topic for free text and regex customer responses .

Autolearning beta for dialog removed

As of this release, the *autolearning* beta has been removed from the **Analytics** section in dialog. You can use actions to test a new and improved autolearning beta. For more information, see Using autolearning to improve assistant responses.

7 April 2023

Never return choice when customer changes topics

If a customer changes a topic during a conversation, there might be some situations when you might not want them to return to the previous action. If you need to do this, a new **Never return** choice is available in **Action settings**. For more information, see Disabling returning to the original topic .

22 March 2023

Autolearning

Autolearning is a beta feature that is available for evaluation and testing purposes in English-language assistants only. Use autolearning to enable your assistant to learn from interactions with your customers and improve responses. For more information, see Using autolearning to improve assistant responses.

21 March 2023

Response modes

Response modes is a beta feature that is available for evaluation and testing purposes. Choose a default *response mode*, which modifies the assistant's behavior when asking clarifying questions. For more information, see Response modes.

20 March 2023

Auto-save setting removed from Global Settings

The **Auto-save** setting was removed from Global Settings. It was a user preference that allowed you to disable automatic saving of actions and applied to all assistants in your instance. Changing the setting applied only to you and didn't affect other users. Actions continue to be automatically saved as you work. For more information, see Saving your actions .

16 March 2023

New algorithm version **Latest (20 Dec 2022)** provides improved irrelevance detection

A new algorithm version is available. The **Latest (20 Dec 2022)** version includes a new irrelevance detection implementation to improve off-topic detection accuracy.

Improvements include:

- Relevant user inputs are expected to get higher confidence, so they are less likely to be considered irrelevant or require clarification
- Irrelevance detection is improved in the presence of direct entity references
- Irrelevance detection is more stable across small changes to input
- Intent detection is more stable regarding occurrence of numerics, such as postal codes
- For German-language assistants, intent detection is more robust in the presence of umlauts

This algorithm was first introduced as the **Beta** version in June 2022. Since then, support for more languages has been added. This algorithm version was stabilized in December 2022 with minor enhancements since that time.

With this new release, the June 1, 2022 version is now labeled as **Previous (01 Jun 2022)**. The oldest release labeled as **01 Jan 2022** is no longer available for training. As of now, the new **Beta** version has the same behavior as the **Latest (20 Dec 2022)** version. Updates to the **Beta** version will be released soon.

For more information, see Algorithm version and training.

10 March 2023

Dialog session variables now available in Preview

If you are using dialog in watsonx Assistant, you can now see session variables for dialog when debugging in Preview. For more information, see Variable values in Preview.

6 March 2023

Improvements to algorithm version beta

Improvements to the current *Beta* algorithm version include:

- Relevant examples are expected to get higher confidence
- For Spanish-language assistants, intent detection is improved in the presence of direct entity references
- Intent detection is more stable regarding occurrence of numerics, such as postal codes
- Intent detection now accounts for fuzzy closed entity mentions
- For German-language assistants, intent detection is more robust in the presence of umlauts

For more information, see Algorithm version and training.

3 March 2023

Adding and using multiple environments

Each assistant has a draft and live environment. For Enterprise plans, you can now add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments. For more information, see Adding and using multiple environments.

Confirmation to return to previous action

If a customer digresses and changes to a new topic, assistants now ask a "yes or no" confirmation question that the customers want to return to the previous action. Previously, the assistant returned to the previous action without asking. New assistants are set to use this confirmation by default. For more information, see Confirmation to return to previous topic.

1 March 2023

Unrecognized requests group names

This change improves the group names for unrecognized requests. For groups with examples phrased in the form of question, the group name can be more indicative of a question rather than a request. For more information, see Use unrecognized requests to get action recommendations.

23 February 2023

Private variables excluded from logs

Private context variables are no longer saved in logs or sent to external services using log webhooks. Private variables are any values stored inside the following objects:

- `context.integrations.*.private` (accessible from actions as `system_integrations.*.private`)
- `context.integrations.*.$private`
- `context.skills.*.user_defined.private`
- `context.skills.*.user_defined.$private`
- `context.private`
- `context.$private`

16 February 2023

Improvements to setting variable values

When you use **Set variable values** on an action step:

- The available choices now match by type. For example, if you want to set a date variable, the choices are limited to other date variables. Previously, all variables of all types were listed as choices.
- You can set a scalar value for each variable type. For example, you can set a specific date for a date variable or set a specific number for a number variable.

For more information, see Storing a value in a session variable.

Confirmation and free text response types setting default

The *Confirmation* and *Free text* response types are now set to **Always ask for this information** by default. For more information, see Skipping steps, always asking steps, or never asking steps.

13 February 2023

Response variations

In actions, you can add *response variations* so that your assistant can respond to the same request in different ways. You can choose to rotate through the response variations sequentially or in random order. For more information, see Adding variations.

Microsoft Teams integration

A Microsoft Teams integration is now available to connect your assistant with the people, content, and tools that your business or community needs to chat, call, and collaborate.

3 February 2023

Action conditions (beta)

An action condition is a boolean test, based on some runtime value; the action executes only if the test evaluates as true. This test can be applied to any variable. By defining action conditions, you can do things such as control user access to actions or create date-specific actions. This is a beta feature that is available for evaluation and testing purposes. For more information, see Adding conditions to an action.

2 February 2023

Detect trigger words (beta)

Use the **Trigger word detected** action to add words or phrases to two separate groups. The first group connects customers with an agent, when it's important for a customer to speak with a live agent rather than activate any further actions. The second group shows customers a customizable warning message, used to discourage customers from interacting with your assistant in unacceptable ways, such as using profanity. This action is included with all new assistants created as of this date. This is a beta feature that is available for evaluation and testing purposes. For more information, see Detecting trigger words.

Changes to unrecognized requests algorithm

In **Analyze**, the **Recognition** page lets you view groups of similar unrecognized requests. You can use the requests as example phrases in new or existing actions to address questions and issues that aren't being answered by your assistant. With this release, the criteria for grouping the requests is relaxed for customers with lesser amounts of data. Also, the group names have been improved with better grammar and to be more representative of the requests. For more information, see Use unrecognized requests to get action recommendations.

1 February 2023

Actions templates updated with new design and new choices

The actions template catalog has a new design that lets you select multiple templates at the same time. It also has new and updated templates, including starter kits you can use with external services such as Google and HubSpot. For more information, see Building actions from a template.

26 January 2023

Display formats for variables

In **Global settings** for actions, **Display formats** lets you specify the display formats for variables that use date, time, numbers, currency, or percentages. You can also choose a default locale to use if one isn't provided by the client application. This lets you make sure that the format of a variable that's displayed in the web chat is what you want for your assistant. For example, you can choose to have the output of a time variable appear in HH:MM format instead of HH:MM:SS. For more information, see Display formats.

Intent recommendations and intent user example recommendations discontinued

As of this release, intent recommendations and intent user example recommendations are discontinued in dialog skills in the classic experience. **Intent recommendations** has been removed from the **Intents** page and **Recommended examples** has been removed from intents. You can use unrecognized requests to get action recommendations.

18 January 2023

Algorithm version stability improvement

As of this date, the **Latest (01 Jun 2022)** and **Beta** algorithm versions now have more stable behavior across retrained models, in the presence of overlapping entities (the same entity value belonging to more than one entity type). Previously, when there were overlapping entities definitions, confidences could differ across different retraining. With this improvement, you can expect to see similar confidences. For more information, see Algorithm version and training.

12 January 2023

Autocorrection setting for actions

The **Global settings** for actions now include an **Autocorrection** setting. *Autocorrection* fixes misspellings that users make in their requests. The corrected words are used to match to an action.

While the setting is new, the autocorrection feature was already used automatically by all English-language assistants. Autocorrection is also available in French-language assistants, but is disabled by default. The autocorrection setting isn't available for any other languages. The new setting lets you disable or enable autocorrection if necessary.

For more information, see Autocorrecting user input.

Improved experience when setting a variable value

The dropdown list for setting a variable value within an action step has a new organization. The new list is intended to provide an improved experience.

11 January 2023

Algorithm version 01-Jun-2022 uses enhanced intent detection by default

As of this date, the algorithm version **Latest (01-Jun-2022)** now uses enhanced intent detection by default. Before this change, some skills that did not include a specific algorithm version selection inadvertently used **Previous (01-Jan-2022)**. You can notice small changes in intent detection behavior when changes are made to an assistant that previously didn't have enhanced intent detection enabled. For more information, see Algorithm version and training.

6 December 2022

Updated expression methods

The following new and updated methods are available in expressions:

- The `Array.joinToArray()` method now supports a new boolean parameter you can use to specify that the data type of values from the input array should be preserved in the returned array.
- The new `String.toJson()` method parses a string containing JSON data and returns a JSON object or array. This method is supported in both actions and dialog.

5 December 2022

Unsupported HTML removed from text responses in channel integrations

HTML tags (except for links) are now automatically removed from text responses that are sent to the Facebook, WhatsApp, and Slack integrations, because those channels do not support HTML formatting. HTML tags are still handled appropriately in channels that support them (such as the web chat) and stored in the session history.

2 December 2022

Pause response type

Use a *Pause* response to have your assistant wait for a specified interval before displaying the next response. This pause might be to allow time for a request to complete, or simply to mimic the appearance of a live agent who might pause between responses. The pause can be of any duration from 1 to 10 seconds. For more information, see Pause response.

17 November 2022

Use unrecognized requests to get action recommendations

In **Analyze**, the new **Recognition** page lets you view groups of similar unrecognized requests. You can use the requests as example phrases in new or existing actions to address questions and issues that aren't being answered by your assistant. For more information, see Use unrecognized requests to get action recommendations.

15 November 2022

Journeys

Beginning with web chat version 6.9.0, you can now create *journeys* to guide your customers through tasks they can already complete on your website. A journey is an interactive, multipart response that can combine text, video, and images, presented in sequence in a small window superimposed over your website.

Journeys are available as a beta feature. For more information, see Guiding customers with journeys.

10 November 2022

Dynamic options

Within the options customer response, you can use the **dynamic** setting to generate the list when you need to ask questions that are potentially different each time and for each customer. You need to set up a list variable as the source of the options. For more information, see Dynamic options.

Extension inspector

You can use the new extension inspector in the action editor **Preview** pane to debug problems with custom extensions. The extension inspector shows detailed information about what data is being sent to and returned from an external API. For more information, see Debugging failures.

3 November 2022

Never ask a step

There may be some situations where you need a step to never ask a question because you anticipate there might be redundant questions in the conversation. A new setting, **Never ask**, is now available for any step that expects a customer response. For more information, see Skipping steps, always asking steps, or never asking steps.

Action notes

You can now add free-form notes to each action. Within each action, you can use **Action notes** to add a description, documentation, comments, or any other annotations to help you keep track of your work as you build an action. For more information, see Using the action editor.

Variable values in Preview

Viewing action variables in Preview has been improved. Now you can see the history of all action variables, rather than one action at a time. For more information, see Variable values.

21 October 2022

Algorithm version updates

The algorithm version setting for both actions and dialog now includes three choices: beta, latest, and previous. For more information, see Algorithm version and training.

12 October 2022

`now(String timezone)` method output includes time zone offset

The string returned from the `now(String timezone)` method now includes the time zone offset (such as `-05:00`). The new format is `yyyy-MM-dd HH:mm:ss 'GMT'XXX` (where `XXX` represents the time zone offset). This change enables accurate time zone computations when used with other date and time methods such as `before`, `after`, and `reformatDateTime`.

If you have an existing action or dialog that depends on the previous format, you can adapt it by reformatting the output using `now(timezone).reformatDateTime('yyyy-MM-dd HH:mm:ss')`.

For more information, see Expression language methods for actions.

23 September 2022

Upload an image as a preview background

On the **Preview** page, you can now upload an image of your organization's website as a background. For more information, see Previewing and sharing your assistant.

16 September 2022

Session ID information on Analyze page

Session ID information for conversations is now displayed on the Conversations tab of the Analyze page. You can also filter customer conversation data by the session ID. From the Conversations tab of the Analyze page, use the Keyword filter to search by session ID. For more information, see Filtering conversations.

The ability to filter on session ID has limited support for conversations that occurred before this feature release. For all conversations that occurred before 16 September 2022, you can filter only by a single session ID at a time.

12 September 2022

Fix for fuzzy matching in German

In some cases, German closed entities were incorrectly matching shorter values over longer values. For example, suppose that you defined two entity values, `Pflege` and `Pflegegeld` in one entity. If a customer accidentally input `Pflegegelb`, the assistant would incorrectly match with `Pflege` rather than `Pflegegeld`.

With this fix, the `Pflegegelb` input value would correctly match `Pflegegeld`, not `Pflege`. For more information, see How fuzzy matching works.

9 September 2022

New operators available for building conditions

Several new operators are available for building conditions in your actions. The options response type now has the `is any of` and `is none of` operators available. For more information, see Operators.

Copy actions to other assistants

You can copy an action from one assistant to another. When you copy an action, references to other actions, variables, and saved responses are also copied. For more information, see Copying an action to another assistant.

Filter variables and saved responses by name

You can now find variables and saved responses more easily. On the Actions page, you can filter variables you created or saved responses you added. Click the search icon, then enter a search string. Your list of variable or saved responses filters to match what you enter.

1 September 2022

Conditioning on days of the week

You can now condition a step on days of the week. This feature is available with the *date* response type and the *Current date* built-in variable.

For example, you might define a customer response in step 1 with the date response type. When the customer responds to that step, they choose a date. You can then condition a later step on whether the date that the customer chose is Wednesday.

New operators available for building conditions

Several new operators are available for building conditions in your actions. The free text response type now has the `contains`, `does not contain`, `matches`, and `does not match` operators available. For more information, see Operators.

Extensions support for arrays

Custom extensions now support passing arrays as parameters and accessing arrays in response variables. For more information, see Calling a custom extension.

26 August 2022

New filter on the Analyze page

You can now filter customer conversation data by the *Greet customer* system action. From the Conversations tab of the Analyze page, open the Actions filter and select **Greet customer**. For more information, see Filtering conversations.

Filter actions by name

You can now find actions more easily. On the Actions page, you can filter actions by name. Click the search icon, then enter a search string. Your list of actions filters to match what you enter.

12 August 2022

Actions templates

When creating actions, you can choose a template that relates to the problem you're trying to solve. Templates help tailor your actions to include items specific to your business need. The examples in each template can also help you to learn how actions work. Actions templates include features such as intents, entities, condition-based responses, synonyms, response validations, and agent fallback. For more information, see Building actions from a template.

Channel name variable

The `Channel name` integration variable lets you add step conditions using these channels: web chat, phone, SMS, WhatsApp, Slack, or Facebook Messenger. For more information, see Adding conditions to a step.

11 August 2022

Algorithm version options available in more languages

Algorithm version options are now available in Arabic, Czech, and Dutch. This allows you to choose which watsonx Assistant algorithm to apply to your future trainings. For more information, see Algorithm version and training.

9 August 2022

New API methods

The v2 API now supports new **Environments** and **Releases** methods:

- **Environments**: Retrieve information about the environments associated with an assistant.
- **Releases**: Retrieve information about the releases (versions) that have been published for an assistant, and assign an available release to an environment.

For more information, see the v2 API Reference.

5 August 2022

Initial value of session variables

You can now set the initial value of a session variable to an expression. For more information, see Creating a session variable.

Uploading intents

If you created intents in the classic experience, you can migrate your intents to actions in watsonx Assistant. This can provide a helpful starting point when you are ready to start building actions. For more information, see Uploading intents as actions.

19 July 2022

Changes to publishing and environments

You can now publish versions of your content without assigning to the live environment, allowing you to make continuous updates before customers see it in production. Also, the formerly separate pages for your draft and live environments now appear as tabs on a single *Environments* page, from which you can set up unique configurations for building and testing in the draft environment, and for your customers in the live environment. For more information, see the Publishing overview.

Logs reader role

Identity and Access Management now includes a new service role, **Logs Reader**, which lets you grant access to Analytics without assigning the Manager role. Use Logs Reader in combination with the Reader or Writer role to provide access to the Analytics page. For more information, see Managing access.

15 July 2022

Segment extension

The Segment extension is now available for Enterprise plans. With this extension, you can use Segment to capture and centralize data about your customers' behavior, including their interactions with your assistant. For more information, see Sending events to Segment.

New expression property

You can now use the `.literal` property to return the exact response that a customer specifies. This property is helpful if a customer uses a synonym of an option, and you want your assistant to respond with the exact phrase they specified. To set this property, click **Set variable values** and assign a session variable to the step variable. Add the `.literal` property to the step variable. Use the session variable in the assistant's response to display the customer's input.

For example, suppose you have an option called `plant` that has `fern` as a synonym. A customer might say `buy a fern`. In this case, you can use the `.literal` property so the assistant's response uses the customer's input. Your assistant might respond, `Great! I see you want to buy a fern.`

11 July 2022

Ability to duplicate an action

You can duplicate an action to reuse information in a new action. When you duplicate an action, the new action includes everything except example phrases. Click the overflow menu on the action you want and select **Duplicate**.

New demo site

Explore our interactive demo site to learn how watsonx Assistant can be used to build powerful, scalable experiences for your users.

24 June 2022

Algorithm version options available in more languages

Algorithm version options are now available in Chinese (Traditional), Japanese, and Korean. This allows you to choose which watsonx Assistant algorithm to apply to your future trainings. For more information, see Algorithm version and training.

16 June 2022

Algorithm version

Algorithm version allows you to choose which watsonx Assistant algorithm to apply to your future trainings. For more information, see Algorithm version and training.

Algorithm beta version (2022-06-10)

Algorithm beta version (2022-06-10) includes a new irrelevance detection algorithm to improve off-topic detection accuracy. Utterances with similar meanings are expected to have more similar confidences in comparison to previous irrelevance detection algorithms. For example, the training utterance `please suggest route from times square` has 100% confidence at runtime. Currently in IBM Cloud, the utterance `please suggest route from central park` gets a low confidence and could be flagged as irrelevant. With beta version (2022-06-10), the same utterance is expected to be predicted correctly with a ~46% confidence.

3 June 2022

Extensions support for importing API document with unsupported methods

When building a custom extension, you can now import an API document even if it contains operations with required array parameters, which are not supported. The unsupported operations are automatically disabled, but this does not affect other operations. (Previously, the entire API document was rejected if it contained unsupported operations.) For more information, see Building a custom extension.

27 May 2022

Support for custom extensions and dialog in Actions preview panel

You can now view your entire assistant from the **Actions preview** panel, including custom extensions and dialog. This allows you to have a complete view of how an action is working. For more information about previewing actions, see Reviewing and debugging your actions.

19 May 2022

Sign out due to inactivity setting

IBM® watsonx™ Assistant now uses the **Sign out due to inactivity setting** from Identity & Access Management (IAM). IBM Cloud account owners can select the time it takes before an inactive user is signed out and their credentials are required again. The default is 2 hours.

An inactive user will see two messages. The first message alerts them about an upcoming session expiration and provides a choice to renew. If they remain inactive, a second session expiration message appears and they will need to log in again.

For more information, see Setting the sign out due to inactivity duration.

12 May 2022

Ability to upload and download example phrases and upload saved customer responses

You can now upload and download example phrases from **Customer starts with** at the start of an action. This can be useful if you have a large number of example phrases and don't want to define them one by one. For more information, see Adding more examples.

You can also now upload saved customer responses from the **Saved responses** page. For more information, see Uploading saved customer responses.

The ability to upload example phrases and saved customer responses is also helpful if you're using the classic experience and want to migrate your intents and entities to watsonx Assistant. For more information, see Migrating intents and entities.

5 May 2022

Success/failure variable for extensions

Each call to a custom extension now returns a `Ran successfully` response variable, which you can use to check the success or failure of the call. For more information, see Checking success or failure.

28 April 2022

Definitive calculation for abandoned actions

Abandonment is now definitively calculated for your actions. On the **Analytics** page, actions are no longer considered `Ongoing` in the action completion analysis. An action is considered abandoned if it was not completed after 1 hour of inactivity and doesn't meet the criteria for any other incompletion reason (escalated to agent, started a new action, or stuck on a step). This change applies only to actions data after April 26, 2022. For more information about action incompletion, see Reasons for incompletion.

Managing operations in extensions

When you add a custom extension to an assistant, you can now choose which operations and response properties will be available to actions. For more information, see Adding an extension to your assistant.

21 April 2022

Ability to duplicate a step

You can now duplicate a step so you don't have to re-create variable settings and customizations. Duplicating a step is helpful when you need to add a step similar to a previous step, but with minor modifications. For more information about how to duplicate a step, see Duplicating a step.

Markdown supported in action editor

The action editor now supports basic Markdown syntax. As you type, the action editor renders the Markdown so you can see the content as your customers will when they interact with the assistant.

5 April 2022

Dialog feature available

The dialog feature is available. If you have a dialog-based assistant that was built using the classic experience, you can now migrate your dialog skill to watsonx Assistant. For more information, see Migrating to watsonx Assistant.

28 March 2022

New service desk support reference implementation

You can use the reference implementation details to integrate the web chat with the Kustomer service desk.

18 March 2022

Custom extensions

If you need to integrate your assistant with an external service that has a REST API, you can now build a custom extension by importing an OpenAPI document. Your assistant can then send requests to the external service and receive response data it can use in the conversation. For example, you might use an extension to interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions.

For more information about custom extensions, see Building a custom extension and Calling a custom extension.

Confirmation customer response type

The confirmation customer response type is now available. Use this response type when a customer's response must be either Yes or No. For more information, see Confirmation.

Search integration highlights text in browser

Search results in watsonx Assistant include a link. Now, when a customer clicks the link, search results are highlighted in their browser so it’s easier for them to see the relevant content. This feature is supported on Chromium browsers, including Google Chrome and Microsoft Edge.

24 February 2022

Regex customer response type

The regex customer response type is now available. Use this response type to capture a value that must conform to a particular pattern or format, such as an email address or telephone number. For more information, see Regex.

17 February 2022

Adding users from the Manage menu

If you want to collaborate with others on your assistants, you can now quickly add users with Administrator and Manager access from the **Manage** menu in your assistant. For more information, see Adding users from the Manage menu.

Preview page share link

When you use the **Copy link to share** button to share your assistant, the shared assistant now mirrors the Preview page. If you share the assistant with a colleague, they are able to see the assistant with any customizations that you made on the Preview page.

10 February 2022

Links in assistant responses can be configured to open in a new tab

When you build an action, your assistant responses can include links. If you're using web chat, you can now control whether the link opens in a new tab. To enable a link to open in a new tab, select **Open link in new tab** from the **Insert link** configuration window. For more information, see Adding assistant responses.

9 February 2022

All instances now default to new experience

All new instances of watsonx Assistant now direct users to the new product experience by default.

IBM® watsonx™ Assistant has been completely overhauled to simplify the end-to-end process of building and deploying a virtual assistant, reducing time to launch and enabling nontechnical authors to create virtual assistants without involving developers. For more information about watsonx Assistant, and instructions for switching between experiences, see Welcome to watsonx Assistant.

If you would like to send us feedback on watsonx Assistant, please use this form.

3 February 2022

Customize the Preview page background

You can now change the background of the **Preview** page to one of your organization's web pages so you can preview and test your assistant from a customer's perspective. For more information, see Previewing and sharing your assistant.

Add a type to session variables

When you create a session variable, you can now assign a type to the variable. For more information, see Creating a session variable. After a type is assigned to a variable, you can set more explicit conditions on that variable. Previously, you were able to check only whether session variables were `defined` or `not defined`. With variable types, you can create conditions based on the type of the variable (for example, `account balance < 100` or `departure date is after today`). For more information, see Operators.

Create saved customer responses

You can now create saved customer responses. There might be some questions that your assistant needs to ask in different steps and actions. For example, a banking assistant might have different actions that ask for a customer's account number. Instead of building the same response over and over, you can create a saved customer response and reuse it across steps in multiple actions. For more information, see Saving and reusing customer responses.

13 January 2022

New setting for options customer response type

In actions, a new **List options** setting allows you to enable or disable the options customer response from appearing in a list. This can be useful to prevent a phone integration from reading a long list of options to the customer. As part of this change, all customer response types now have a **Settings** icon. **Allow skipping** has moved from **Edit Response** and is now found in the new settings. For more information, see Collecting information from your customers.

24 December 2021

Apache Log4j security vulnerability updates

IBM® watsonx™ Assistant upgraded to using Log4j version 2.17.0, which addresses all of the Critical severity and High severity Log4j CVEs, specifically CVE-2021-45105, CVE-2021-45046, and CVE-2021-44228.

3 December 2021

Configure webhook timeout

From the **Pre-message webhook** and **Post-message webhook** configuration pages, you can configure the webhook timeout length from a minimum of 1 second to a maximum of 30 seconds. For more information, see Extending your assistant with webhooks.

User-based switching between watsonx Assistant experiences

Previously, switching between watsonx Assistant and the classic experience was instance-based. For example, if a user switched from the classic experience to watsonx Assistant, all users of that instance were switched to watsonx Assistant. Now, switching between the experiences is user-based. So, any user of an instance can switch between the new and classic experiences, and other users of that instance are not affected.

27 November 2021

New API version

The current API version is now `2021-11-27`. This version introduces the following changes:

- The `output.text` object is no longer returned in `message` responses. All responses, including text responses, are returned only in the `output.generic` array.

12 November 2021

Completion analytic information

On the **Analyze** page, the **How often** chart can now also show the percentage of complete actions. Use the icon in the upper right of the chart to toggle between a line chart that shows the percentage of complete actions and a bar chart that shows the number of complete and incomplete actions. For more information, see Improving completion.

Preview page update

The **Test integrations** panel no longer exists on the **Preview** page. You can manage your draft web chat channel from the **Preview** page. However, all other draft environment integrations are managed from the **Draft environment** page. For more information, see Previewing and sharing your assistant.

4 November 2021

Draft and Live Environment pages

Two pages, **Draft environment** and **Live environment**, help you to see how your channels and resolution methods are connected, both for testing/preview and for live deployment. The Draft environment page is new as of this release. The Live environment page was previously named Connect. For more information, see Overview: Publishing and deploying your assistant.

1 November 2021

New API version

The current API version is now `2021-11-01`. This version includes the following changes:

- Currency and percentage values are stored as `system_type` objects in actions. Previously, they were stored as atomic numbers.
- The `$skip_user_input` flag is now in `contex.global.system.skip_user_input`. Previously, it was in `context.skills.["skill-reference"].user_defined.skip_user_input`.
- When an expression is evaluated, the result is no longer subject to a SpEL evaluation. As a result, actions no longer support inline SpEL expressions such as `<? code ?>` or variable references such as `<? ${variable_name} ?>`. These are all superseded by the rich text editor's

validated JSON serialization. Previously, you could write dynamic SpEL expressions such as `<? 1+1 ?>` to a scalar value and expect the result to be `2`. Now the corresponding expression construct must be used, for example, `"expression": "1+1"`.

Add variables to links

When including a link in an assistant response, you can now access and use variables. In the URL field for a link, type a dollar sign (`$`) character to see a list of variables to choose from.

25 October 2021

Facebook and Slack integrations now available

IBM® watsonx™ Assistant now includes integrations for Facebook Messenger and Slack.

Analytics for draft and live environments

The **Analyze** page now lets you see analytics data for either the draft or live environments. For more information, see Use analytics to review your entire assistant at a glance.

7 October 2021

Introducing watsonx Assistant

This new experience, focused on using **actions** to build customer conversations, is designed to make it simple enough for *anyone* to build a virtual assistant. Building, testing, publishing, and analyzing your assistant can all now be done in one simple and intuitive interface.

- New **navigation** provides a workflow for building, previewing, publishing, and analyzing your assistant.
- Each assistant has a **home page** with a task list to help you get started.
- Build conversations with **actions**, which represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer.
- A new way to **publish** lets you review and debug your work in a draft environment before going live to your customers.
- Use a new suite of **analytics** to improve your assistant. Review which actions are being completed to see what your customers want help with, determine if your assistant understands and addresses customer needs, and decide how can you make your assistant better.
- New Top intents and top entities.

16 September 2021

Enhanced intent detection for French, Italian, and Spanish dialog skills

The new intent detection model improves your assistant's ability to understand what customers want. This model is now available in dialog skills using French, Italian, and Spanish.

Change to the irrelevance detection option

As of this release, new English dialog skills no longer include the option to choose between the **Enhanced** or **Existing** irrelevance detection. By default, intent detection and irrelevance detection are paired like this:

- If you use the dialog skill options to choose enhanced intent detection, it is automatically paired with enhanced irrelevance detection.
- If you use the dialog skill options to choose existing intent detection, it is automatically paired with existing irrelevance detection.

For more information, see Defining what's irrelevant.

If necessary, you can use the Update workspace API to set your English-language assistant to one of the four combinations of intent and irrelevance detection:

- Enhanced intent recognition and enhanced irrelevance detection
- Enhanced intent recognition and existing irrelevance detection
- Existing intent recognition and enhanced irrelevance detection

- Existing intent recognition and existing irrelevance detection

For French, Italian, and Spanish, you can use the API to set your assistant to these combinations:

- Enhanced intent recognition and enhanced irrelevance detection
- Existing intent recognition and existing irrelevance detection

15 September 2021

Dialog skill "Try it out" improvements

The **Try it out** pane now includes these changes:

- It now includes runtime warnings in addition to runtime errors.
- For dialog skills, the **Try it out** pane now uses the React UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update was enabled on these dates:
  - September 9, 2021 in the Tokyo and Seoul data centers
  - September 13, 2021 in the London, Sydney, and Washington, D.C. data centers
  - September 15, 2021 in the Dallas and Frankfurt data centers

13 September 2021

Dialog skill "Try it out" improvements

For dialog skills, the **Try it out** pane now uses the React UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update was enabled on September 9, 2021 in the Tokyo and Seoul data centers. On September 13, 2021, the update was enabled in the London, Sydney, and Washington, D.C. data centers.

Disambiguation feature updates

The dialog skill disambiguation feature now includes improved features:

- **Increased control**: The frequency and depth of disambiguation can now be controlled by using the **sensitivity** parameter in the workspace API. There are 5 levels of sensitivity:
  - `high`
  - `medium_high`
  - `medium`
  - `medium_low`
  - `low`

  The default (`auto`) is `medium_high` if this option is not set.
- **More predictable**: The new disambiguation feature is more stable and predictable. The choices shown may sometimes vary slightly to enable learning and analytics, but the order and depth of disambiguation is largely stable.

These new features may affect various metrics, such as disambiguation rate and click rates, as well as influence conversation-level key performance indicators such as containment.

If the new disambiguation algorithm works differently than expected for your assistant, you can adjust it using the sensitivity parameter in the update workspace API. For more information, see Update workspace.

9 September 2021

Actions skill improvements

Actions skills now include these new features:

- **Change conversation topic**: In general, an action is designed to lead a customer through a particular process without any interruptions. In real life, however, conversations almost never follow such a simple flow. In the middle of a conversation, customers might get distracted, ask questions about related issues, misunderstand something, or just change their minds about what they want to do. The **Change conversation topic** feature enables your assistant to handle these digressions, dynamically responding to the user by changing the conversation topic as needed. For more information, see Allowing your customers to change the topic of the conversation .
- **Fallback action**: The built-in action, *Fallback*, provides a way to automatically connect customers to a human agent if they need more help. This action helps you to handle errors in the conversation, and is triggered by these conditions:
  - Step validation failed: The customer repeatedly gave answers that were not valid for the expected customer response type.
  - Agent requested: The customer directly asked to be connected to a human agent.
  - No action matches: The customer repeatedly made requests or asked questions that the assistant did not understand.

  For more information, see Editing the fallback action .

Dialog skill "Try it out" improvements

For dialog skills, the **Try it out** pane now uses the React UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the "Try it out" pane. This update will be implemented incrementally, starting with service instances in the Tokyo and Seoul data centers.

2 September 2021

Deploy your assistant on the phone in minutes

We have partnered with IntelePeer to enable you to generate a phone number for free within the phone integration. Simply choose to generate a free number when following the prompts to create a phone integration, finish the setup, and a number is assigned to your assistant. These numbers are robust and ready for production.

Connect to your existing service desks

We have added step-by-step documentation for connecting to Genesys and Twilio Flex over the phone. Easily hand off to your live agents when your customers require telephony support from your service team. Watson Assistant deploys on the phone via SIP, so most phone based service desks can easily be integrated via SIP trunking standards.

23 August 2021

Intent detection updates

Intent detection for the English language has been updated with the addition of new word-piece algorithms. These algorithms improve tolerance for out-of-vocabulary words and misspelling. This change affects only English-language assistants, and only if the enhanced intent recognition model is enabled.

Automatic retraining of old skills and workspaces

As of August 23, 2021, Watson Assistant enabled automatic retraining of existing skills in order to take advantage of updated algorithms. The Watson Assistant service will continually monitor all ML models, and will automatically retrain those models that have not been retrained within the previous 6 months. For more information, see Automatic retraining.

19 August 2021

Actions preview now includes debug mode and variable values

When previewing your actions, you can use **debug mode** and **variable values** to ensure your assistant is working the way you expect.

Debug mode allows you to go to the corresponding step by clicking on a step locator next to each message. It shows you the confidence score of top three possible action when the input triggers an action. You can also follow the step in the action editor along with the conversation flow.

Variable values shows you a list of the variables and their values of current action and the session variables. You can check and edit variables during the conversation flow.

17 August 2021

New service desk support reference implementation

You can use the reference implementation details to integrate the web chat with the Oracle B2C Service service desk.

29 July 2021

Salesforce and Zendesk deployment changes

The Salesforce and Zendesk integrations have been updated to use the new chat history widget. The updated deployment process applies to all new deployments, including any redeployments of existing Salesforce and Zendesk connections. However, existing deployments are not affected and do not need to be modified or redeployed at this time.

Fallback value for session variables

In action skills, you can now set a fallback value for session variables. This feature lets you to define a value for a session variable if a user-defined value isn't found. To learn more, see Creating a session variable.

16 July 2021

Logging API changes

The internal storage and processing of logs has changed. Some undocumented fields or filters might no longer be available. (Undocumented features are not officially supported and might change without notice.)

New API version

The current API version (v1 and v2) is now `2021-06-14`. The following changes were made with this version:

- The `metadata` property of entities detected at run time is deprecated. For detailed information about detected system entities, see the `interpretation` property.
- The data types of certain entity mentions are no longer automatically converted:
  - Numbers in scientific notation (such as `1E10`), which were previously converted to numbers
  - Boolean values (such as `false`), which were previously converted to booleans

These values are now returned as strings.

17 June 2021

Actions skill now generally available

As of this release, the beta program has ended, and actions skills are available for general use.

An actions skill contains actions that represent the tasks you want your assistant to help your customers with. Each action contains a series of steps that represent individual exchanges with a customer. Building the conversation that your assistant has with your customers is fundamentally about deciding which steps, or which user interactions, are required to complete an action. After you identify the list of steps, you can then focus on writing engaging content to turn each interaction into a positive experience for your customer. For more information, see Overview: Editing actions.

Date and time response types

New to action skills, these response types allow you to collect date and time information from customers as they answer questions or make requests. For more information, see Collecting information from your customers.

New built-in variables

Two kinds of built-in variables are now available for action skills.

- **Set by assistant** variables include the common and essential variables `Now`, `Current time`, and `Current date`.
- **Set by integration** variables are `Timezone` and `Locale` and are available to use when connected to a webhook or integration.

For more information, see Using variables to manage conversation information.

Universal language model now generally available

You now can build an assistant in any language you want to support. If a dedicated language model is not available for your target language, create a skill that uses the universal language model. The universal model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from training data written in the target language that you add to it. For more information, see Understanding the universal language model.

3 June 2021

Log webhook support for actions and search skills

The log webhook now supports messages exchanged with actions skills and search skills, in addition to dialog skills. For more information, see Logging activity with a webhook.

27 May 2021

Change to conversation skill choices

When adding skills to new or existing assistant, the conversation skill choices have been combined, so that you pick from either an actions skill or a dialog skill.

With this change:

- New assistants can use up to two skills, either actions and search or dialog and search. Previously, new assistants could use up to three skills: actions, dialog, and search.
- Existing assistants that already use an actions skill and a dialog skill together can continue to use both.
- The ability to use actions and dialog skills together in a new assistant is planned for 2H 2021.

20 May 2021

Actions skill improvement

Actions now include a new choice, **Go to a subaction**, for what to do next in a step. This feature lets you can call one action from another action, to switch the conversation flow to another action to perform a certain task. If you have a portion of an action that can be applied across multiple use cases you can build it once and call to it from each action. This new option is available in the **And then** section of each step. For more information, see Go to a subaction.

21 April 2021

Preview button for testing your assistant

For testing your assistant, the new Preview button replaces the previous Preview tile in Integrations.

New checklist with steps to go live

Each assistant includes a checklist that you can use to ensure you're ready to go live.

Actions skill improvement

Actions now include currency and percentage response types.

Learn what's new

The *What's new* choice on the help menu opens a list of highlighting recent features.

14 April 2021

Actions skill improvement

Actions now include a free text response type, allowing you to capture special instructions or requests that a customer wants to pass along.

8 April 2021

Deploy your assistant to WhatsApp - now generally available

Make your assistant available through WhatsApp messaging so it can exchange messages with your customers where they are. This integration, which is now generally available, creates a connection between your assistant and WhatsApp by using Twilio as a provider. For more information, see Integrating with WhatsApp.

Web chat home screen now generally available

Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to easily start chatting with the assistant. For more information about the home screen feature, see Configuring the home screen. The home screen feature is now enabled by default for all new web chat deployments. Also, you can now access context variables from the home screen. Note that initial context must be set using a `conversation_start` node. For more information, see Starting the conversation.

Connect to human agent response type allows more text

In a dialog skill, the response type *Connect to human agent* now allows 320 characters in the *Response when agents are online* and *Response when no agents are online* fields. The previous limit was 100 characters.

Legacy system entities deprecated

In January 2020, a new version of the system entities was introduced. As of April 2021, only the new version of the system entities is supported for all languages. The option to switch to using the legacy version is no longer available.

6 April 2021

Service API endpoint change

As explained in December 2019, as part of work done to fully support IAM authentication, the endpoint you use to access your Watson Assistant service programmatically is changing. The old endpoint URLs are deprecated and **will be retired on 26 May 2021**. Update your API calls to use the new URLs.

The pattern for the endpoint URL changes from `gateway-{location}.watsonplatform.net/assistant/api/` to `api.{location}.assistant.watson.cloud.ibm.com/`. The domain, location, and offering identifier are different in the new endpoint. For more information, see Updating endpoint URLs from watsonplatform.net.

- If your service instance API credentials show the old endpoint, create a new credential and start using it today. After you update your custom applications to use the new credential, you can delete the old one.
- For a web chat integration, you might need to take action depending on when and how you created your integration.
    - If you tied your deployment to a specific web chat version by using the `clientVersion` parameter and specified a version earlier than version 3.3.0, update the parameter value to use version 3.3.0 or later. Web chat integrations that use the latest or 3.3.0 and later versions will not be impacted by the endpoint deprecation.

    - If you created your web chat integration before May 2020, check the code snippet that you embedded in your web page to see if it refers to `watsonplatform.net`. If so, you must edit the code snippet to use the new URL syntax. For example, change the following URL:

      ```
      $ <script src="https://assistant-web.watsonplatform.net/loadWatsonAssistantChat.js"></script>
      ```

      The correct syntax to use for the source service URL looks like this:

      ```
      $ src="https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js"
      ```

- If your web chat integration connects to a Salesforce service desk, then you must edit the API call that is included in the code snippet that you added to the Visualforce Page that you created in Salesforce. From Salesforce, search for *Visualforce Pages*, and find your page. In the `<iframe>` snippet that you pasted into the page, make the following change:

  Replace: `src="https://assistant-integrations-{location}.watsonplatform.net/public/salesforceweb"` with a url with this syntax:

  ```
  src="https://integrations.{location}.assistant.watson.appdomain.cloud/public/salesforceweb/{integration-id}/agent_application?version=2020-09-24"`
  ```

  From the Web chat integration Salesforce live agent setup page, find the *Visualforce page markup* field. Look for the `src` parameter in the `<iframe>` element. It contains the full URL to use, including the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Slack integration that is over 7 months old, make sure the Request URL is using the proper endpoint.
    - Go to the Slack API web page. Click *Your Apps* to find your assistant app. Click *Event Subscriptions* from the navigation pane.
    - Edit the Request URL.

  For example, if the URL has the syntax: `https://assistant-slack-{location}.watsonplatform.net/public/message`, change it to have this syntax:

  ```
  https://integrations.{location}.assistant.watson.appdomain.cloud/public/slack/{integration-id}/message?version=2020-09-24
  ```

  Check the *Generated request URL* field in the Slack integration setup page for the full URL to use, which includes the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Facebook Messenger integration that is over 7 months old, make sure the Callback URL is using the proper endpoint.
    - Go to the Facebook for Developers web page.
    - Open your app, and then select *Messenger>Settings* from the navigation pane.
    - Scroll down to the *Webhooks* section and edit the *Callback URL* field.

      For example, if the URL has the syntax: `https://assistant-facebook-{location}.watsonplatform.net/public/message/`, change it to have this syntax:

      ```
      https://integrations.{location}.assistant.watson.appdomain.cloud/public/facebook/{integration-id}/message?version=2020-09-24
      ```

      Check the *Generated callback URL* field in the Facebook Messenger integration setup page for the full URL to use, which includes the appropriate `{location}` and `{integration-id}` values for your instance.

- For a Phone integration, if you connect to existing speech service instances, make sure those speech services use credentials that were generated with the latest endpoint syntax (a URL that starts with `https://api.{location}.speech-to-text.watson.cloud.ibm.com/`).
- For a search skill, if you connect to an existing Discovery service instance, make sure the Discovery service uses credentials that were generated with the supported syntax (a URL that starts with `https://api.{location}.discovery.watson.cloud.ibm.com/`).
- No action is required for the following integration types:
    - Intercom
    - SMS with Twilio
    - WhatsApp with Twilio
    - Zendesk service desk connection from web chat

23 March 2021

Actions skill improvement

Actions have a new toolbar making it easier to send feedback, access settings, save, and close.

17 March 2021

Channel transfer response type

Dialog skills now include a channel transfer response type. If your assistant uses multiple integrations to support different channels for interaction with users, there might be some situations when a customer begins a conversation in one channel but then needs to transfer to a different channel. The most common such situation is transferring a conversation to the web chat integration, to take advantage of web chat features such as service desk integration. For more information, see [Adding a Channel transfer response type](/docs/watson-assistant?topic=watson-assistant-dialog-overview#dialog-overview-add-channel-transfer.

Intercom and WhatsApp integrations now available in Lite plan

The integrations for Intercom and WhatsApp are now available in the Lite plan for Watson Assistant. For more information, see Integrating with Intercom and Integrating with WhatsApp.

16 March 2021

Session history now generally available

Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. It is enabled by default. For more information about this feature, see Session history.

Session history persists within only one browser tab, not across multiple tabs. The dialog provides an option for links to open in a new tab or the same tab. See this example for more information on how to format links to open in the same tab.

Session history saves changes that are made to messages with the pre:receive event so that messages still look the same on rerender. This data is only saved for the length of the session. If you prefer to discard the data, set `event.updateHistory = false;` so the message is rerendered without the changes that were made in the pre:receive event.

instance.updateHistoryUserDefined() provides a way to save state for any message response. With the state saved, a response can be rerendered with the same state. This saved state is available in the `history.user_defined` section of the message response on reload. The data is saved during the user session. When the session expires, the data is discarded.

Two new history events, history:begin and history:end announce the beginning and end of the history of a reloaded session. These events can be used to view the messages that are being reloaded. The history:begin event allows you to edit the messages before they are displayed.

See this example for more information on saving the state of customResponse types in session history.

Channel switching

You can now create a dialog response type to functionally generate a connect-to-agent response within channels other than web chat. If a user is in a channel such as Slack or Facebook, they can trigger a channel transfer response type. The user receives a link that forwards them to your organization's website where a connection to an agent response can be started within web chat. For more information, see Adding a Channel transfer response type.

11 March 2021

Actions skill improvement

Updated the page where you configure a step with an *Options* reply constraint. Now it's clearer that you have a choice to make about whether to always ask for the option value or to skip asking. For more information, see Skipping steps, always asking steps, or never asking steps .

4 March 2021

Support for every language!

You now can build an assistant in any language you want to support. If a dedicated language model is not available for your target language, create a skill that uses the universal language model. The universal model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from training data written in the target language that you add to it.

The universal model is available as a beta feature. For more information, see Understanding the universal language model.

Actions skill improvement

Now you can indicate whether or not to ask for a number when you apply a number reply constraint to a step. Test how changes to this setting might help speed up a customer's interaction. Under the right circumstances, it can be useful to let a number mention be recognized and stored without having to explicitly ask the customer for it. For more information, see Skipping steps, always asking steps, or never asking steps.

1 March 2021

Introducing the *Enterprise* plan!

The Enterprise plan includes all of the market differentiating features of the Plus plan, but with higher capacity limits, additional security features, custom onboarding support to get you going, and a lower overall cost at higher volumes.

To have a dedicated environment provisioned for your business, request the *Enterprise with Data Isolation* plan. To submit a request online, go to Contact WA Enterprise.

The Enterprise plan is replacing the Premium plan. The Premium plan is being retired today. Existing Premium plan users are not impacted. They can continue to work in their Premium instances and create instances up to the 30-instance limit. New users do not see the Premium plan as an option when they create a service instance.

For more information, see the Pricing page.

Other plan changes

Our pricing has been revised to reflect the features we've added that help you build an assistant that functions as a powerful omnichannel SaaS application.

Starting on 1 March 2021, the Plus plan starts at $140 per month and includes your first 1,000 monthly users. You pay $14 for each additional 100 active users per month. Use of the voice capabilities that are provided by the *Phone* integration are available for an additional $9 per 100 users per month.

The Plus Trial plan was renamed to Trial.

SOC 2 compliance

Watson Assistant is SOC 2 Type 2 compliant, so you know your data is secure.

The System and Organization Controls framework, developed by the American Institute of Certified Public Accountants (AICPA), is a standard for controls that protect information stored in the cloud. SOC 2 reports provide details about the nature of internal controls that are implemented to protect customer-owned data. For more information, see IBM Cloud compliance programs.

25 February 2021

Search skill can emphasize the answer

You can configure the search skill to highlight text in the search result passage that Discovery determines to be the exact answer to the customer's question.

Integration changes

The following changes were made to the integrations:

- The name of *Preview link* integration changed to *Preview*.

- The *Web chat* and *Preview* integrations are no longer added automatically to every new assistant.

  The integrations continue to be added to the *My first assistant* that is generated for you automatically when you first create a new service instance.

Message and log webhooks are generally available

The premessage, postmessage, and log webhooks are now generally available. For more information about them, see Webhook overview.

11 February 2021

The `user_id` value is easier to access

The `user_id` property is used for billing purposes. Previously, it was available from the context object as follows:

- v2: `context.global.system.user_id`
- v1: `context.metadata.user_id`

The property is now specified at the root of the `/message` request in addition to the context object. The built-in integrations typically set this property for you. If you're using a custom application and don't specify a `user_id`, the `user_id` is set to the `session_id` (v2) or `conversation_id` (v1) value.

Digression bug fix

Fixed a bug where digression setting changes that were made to a node with slots were not being saved.

5 February 2021

Documentation update

The phone and *SMS with Twilio* deployment documentation was updated to include instructions for migrating from Voice Agent with Watson. For more information, see Migrating from Voice Agent with Watson.

27 January 2021

German language improvements

A word decomposition function was added to the intent and entity recognition models for German-language dialog skills.

A characteristic of the German language is that some words are formed by concatenating separate words to form a single compound word. For example, `festnetznummer` (landline number) concatenates the words `festnetz` (landline) and `nummer` (number). When your customers chat with your assistant, they might write a compound word as a single word, as hyphenated words, or as separate words. Previously, the variants resulted in different intent confidence scores and different entity mention counts based on your training data. With the addition of the word decomposition function, the models now treat all compound word variants as equivalent. This update means you no longer need to add examples of every variant of the compound words to your training data.

19 January 2021

The *Phone* and *SMS with Twilio* integrations are now generally available!

For more information, see:

- Integrating with phone
- Integrating with *SMS with Twilio*

Preview link change

When you create a preview link, you can now test your skill from a chat window that is embedded in the page. You can also copy the URL that is provided, and open it in a web browser to see an IBM-branded web page with the web chat embedded in it. You can share the URL to the public IBM web page with others to get help with testing or for demoing purposes.

Import and export UI changes

The label on buttons for importing skills changed from *Import* to *Upload*, and the label on buttons for exporting skills changed from *Export* to *Download*.

Coverage metric change

The coverage metric now looks for nodes that were processed with a node condition that includes the `anything_else` special condition instead of nodes that are named `Anything else`. For more information, see Starting and ending the dialog.

15 January 2021

Use new webhooks to process messages!

A set of new webhooks is available as a beta feature. You can use the webhooks to perform preprocessing tasks on incoming messages and postprocessing tasks on the corresponding responses. You can use the new log webhook to log each message with an external service. For more information, see Webhook overview.

New service desk support reference implementation

You can use the reference implementation details to integrate the web chat with the NICE inContact service desk. For more information, see Integrating with phone and NICE CXone contact center.

Phone and *SMS with Twilio* integration updates

The phone integration now enables you to specify more than one phone number, and the numbers can be imported from a comma-separated values (CSV) file. The *SMS with Twilio* integration no longer requires you to add your SMS phone number to the setup page.

6 January 2021

Import and export UI changes

The label on buttons for importing intents and entities changed from *Import* to *Upload*. The label on buttons for exporting intents and entities changed from *Export* to *Download*.

4 January 2021

Dialog methods updates

Documentation and examples were added for the following supported dialog methods:

- `JSONArray.addAll(JSONArray)`
- `JSONArray.containsIgnoreCase(value)`
- `String.equals(String)`
- `String.equalsIgnoreCase(String)`

For more information, see Expression language methods for dialog.

17 December 2020

Accessibility improvements

The product was updated to provide enhanced accessibility features.

14 December 2020

Increased Phone and SMS with Twilio integrations availability

These beta SMS and voice capabilities are now available from service instances that are hosted in Seoul, Tokyo, London, and Sydney.

Improved JSON editor

The JSON editor in the dialog skill was updated. The editor now uses JSON syntax highlighting and allows you to expand and collapse objects.

Connect to agent from actions skill

The actions skill now supports transferring a customer to an agent from within an action step. For more information, see Connecting to a live agent.

4 December 2020

Introducing more service desk options for web chat

When you deploy your assistant by using the web chat integration, there are now reference implementations that you can use for the following service desks:

- Twilio Flex
- Genesys Cloud

Alternatively, you can bring your own service desk by using the service desk extension starter kit.

Autolearning has been moved and improved

Go to the *Analytics>Autolearning* page to enable the feature and see visualizations that illustrate how autolearning impacts your assistant's performance over time. For more information, see Using autolearning to improve assistant responses.

Search from actions skill

The actions skill now supports triggering a search that uses your associated search skill from within an action step. For more information, see Search for the answer.

System entities language support change

The new system entities are now used by all skills except Korean-language dialog skills. If you have a Korean skill that uses the older version of the system entities, update it. The legacy version will stop being supported for Korean skills in March 2021.

Disambiguation selection enhancement

When a customer chooses an option from a disambiguation list, the corresponding intent is submitted. With this latest release, a confidence score of 1.0 is assigned to the intent. Previously, the original confidence score of the option was used.

Skill import improvements

Importing of large skills from JSON data is now processed in the background. When you import a JSON file to create a skill, the new skill tile appears immediately. However, depending on the size of the skill, it might not be available for several minutes while the import is being processed. During this time, the skill cannot be opened for editing or added to an assistant, and the skill tile shows the text **Processing**.

23 November 2020

Deploy your assistant to WhatsApp!

Make your assistant available through WhatsApp messaging so it can exchange messages with your customers where they are. This beta integration creates a connection between your assistant and WhatsApp by using Twilio as a provider. For more information, see Integrating with WhatsApp.

13 November 2020

New coverage metric and enhanced intent detection model

The following features are available in service instances hosted in all data center locations except Dallas.

Introducing the coverage metric!

Want a quick way to see how your dialog is doing at responding to customer queries? Enable the new coverage metric to find out. The coverage metric measures the rate at which your dialog is confident that it can address a customer's request per message. For conversations that are not covered, you can review the logs to learn more about what the customer wanted. For the metric to work, you must design your dialog to include an *Anything else* node that is processed when no other dialog node intents are matched. For more information, see Graphs and statistics.

Try out the enhanced intent detection model

The new model, which is being offered as a beta feature in English-language dialog and actions skills, is faster and more accurate. It combines traditional machine learning, transfer learning, and deep learning techniques in a cohesive model that is highly responsive at run time.

3 November 2020

Suggestions are now generally available

The Suggestions feature that is available for the web chat integration is generally available and is enabled by default when you create a new web chat integration. For more information, see Configuring suggestions.

29 October 2020

System entity support changes

For English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills only the new system entities API version is supported. For backward compatibility, both the `interpretation` and `metadata` attributes are included with the recognized entity object. The new system entity version is enabled automatically for dialog skills in the Arabic, Chinese, Korean, and Japanese languages. You can choose to use the legacy version of the system entities API by switching to it from the **Options>System Entities** page. This settings page is not displayed in English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills because use of the legacy version of the API is no longer supported for those languages. For more information about the new system entities, see System entities.

28 October 2020

Introducing the *actions skill*!

The actions skill is the latest step in the continuing evolution of Watson Assistant as a software as a service application. The actions skill is designed to make it simple enough for *anyone* to build a virtual assistant. We've removed the need to navigate between intents, entities, and dialog to create conversational flows. Building can all now be done in one simple and intuitive interface.

The actions skill is available as a beta feature.

Web chat integration is created automatically

When you create a new assistant, a web chat integration is created for you automatically (in addition to the preview link integration, which was created previously). These integrations are added also to the assistant that is auto-generated (named *My first assistant*) when you create a new service instance. For more information, see Web chat overview.

Text messaging integration was renamed

The *Twilio messaging* integration was renamed to *SMS with Twilio*.

9 October 2020

Search skill update

Support was added for a new version of the Discovery API which adds the following capabilities:

- The search skill can now connect to existing Premium Discovery service instances.
- When you connect to a Box, Sharepoint, or Web crawl data collection, the result content fields are automatically populated for you. The **Title** now uses the `title` field from the source document instead of the `extracted_metadata.title` field, which provides better results.

1 October 2020

Introducing the *Phone* integration!

Your customers are calling; now your assistant can answer. Add a phone integration to enable your assistant to answer customer support calls. The integration connects to your existing Session Initiation Protocol (SIP) trunk, which routes incoming calls to your assistant. For more information, see Integrating with phone.

Introducing the *Twilio messaging* integration!

Enable your assistant to receive and respond to questions that customers submit by using SMS text messaging. When you enable both new integrations, your assistant can send text messages to a customer in the context of an ongoing phone conversation. For more information, see Integrating with SMS.

The *Phone* and *Twilio messaging* integrations are available as beta features in Watson Assistant service instances that are hosted in Dallas, Frankfurt, and Washington, DC.

The web chat integration is added to new assistants automatically

Much like the *Preview link* integration, the *Web chat* integration now is added to the *My first assistant* assistant that is created for new users automatically.

24 September 2020

Introducing the containment metric!

Want a quick way to see how often your assistant has to ask for help? Enable the new containment metric to find out. The containment metric measures the rate at which your assistant is able to address a customer's goal without human intervention. For conversations that are not contained, you can review the logs to understand what led customers to seek help outside of the assistant. For the metric to work, you must design your dialog to flag requests for additional support when they occur. For more information, see Graphs and statistics.

Chat transfer improvements

When you add the *Connect to human agent* response type to a dialog node, you can now define messages to show to your customers during the transfer, and can specify service desk agent routing preferences. For more information, see Adding a *Connect to human agent* response type.

22 September 2020

New API version

The current v2 API version is now `2020-09-24`. In this version, the structure of the `search` response type has changed. The `results` property has been removed and replaced with two new properties:

- `primary_results` property includes the search results that should be displayed in the initial response to a user query.
- `additional_results` property includes search results that can be displayed if the user wants to see more.

The search skill configuration determines how many search results are included in the `primary_results` and `additional_results` properties.

Search skill improvements

The following improvements were made to the search skill:

- **Control the number of search results** : You can now customize the number of search results that are shown in a response from the search skill. For more information, see Configure the search.
- **FAQ extraction is available for web crawl data collections** : When you create a web crawl data collection type, you can now enable the FAQ extraction beta feature. FAQ extraction allows the Discovery service to identify question and answer pairs that it finds as it crawls the website.

16 September 2020

Search skill refinement change

The search refinement beta feature that was added in June now is disabled by default. Enable the feature to refine the search results that are returned from the Discovery service. For more information, see Configure the search.

25 August 2020

Give the web chat integration a try!

You can now use the web chat integration with a Lite plan. Previously, the web chat was available to Plus or higher plans only. For more information, see Web chat overview.

12 August 2020

v2 Logs API is available

If you have a Premium plan, you can use the v2 API `logs` method to list log events for an assistant. For more information, see the API reference documentation.

5 August 2020

Enable your skill to improve itself

Try the new **autolearning** beta feature to empower your skill to improve itself automatically over time. Your skill observes customer choices to understand which choices are most often the best. As its confidence grows, your skill presents better options to get the right answers to your customers with fewer clicks. For more information, see Using autolearning to improve assistant responses.

Show more of search results

When search results are returned from the search skill, the customer can now click a twistie to expand the search result card to see more of the returned text.

29 July 2020

The @sys-location and @sys-person system entities were removed

The `@sys-location` and `@sys-person` system entities are no longer listed on the *System entities* page. If your dialog uses one of these entities, a red `Entity not created` notification is displayed to inform you that the entity is not recognized.

Skill menu actions moved

The menu that was displayed in the header of the skill while you were working with a skill was removed. The actions that were available from the menu, such as import and export, are still available. Go to the Skills page, and click the menu on the skill tile.

The import skill process was updated to support overwriting an existing skill on import.

Dialog issues were addressed

These dialog issues were addressed:

- Fixed an issue with adding a jump-to from a conditional response in one node to a conditional response in another node.
- The page now responds better when you scroll horizontally to see multiple levels of child nodes.

15 July 2020

Support ended for @sys-location and @sys-person

The person and location system entities, which were available as a beta feature in English dialog skills only, are no longer supported. You cannot enable them. If your dialog uses them, they are ignored by the service.

Use contextual entities to teach your skill to recognize the context in which such names are used. For more information about contextual entities, see Annotation-based method.

For more information about how to use contextual entities to identify names of people, see the Detecting Names And Locations With Watson Assistant blog post on Medium.

How legacy numeric system entities are processed has changed

All new dialog skills use the new system entities automatically.

For existing skills that use legacy numeric system entities, how the entities are processed now differs based on the skill language.

- Arabic, Chinese, Korean, and Japanese dialog skills that use legacy numeric system entities function the same as before.
- If you choose to continue to use the legacy system entities in European-language dialog skills, a new legacy API format is used. The new legacy API format simulates the legacy system entities behavior. In particular, it returns a `metadata` object and does not stop the service from identifying multiple system entities for the same input string. In addition, it returns an `interpretation` object, which was introduced with the new version of system entities. Review the `interpretation` object to see the useful information that is returned by the new version.

Update your skills to use the new system entities from the **Options>System Entities** page.

Web chat security is generally available

Enable the security feature of web chat so that you can verify that messages sent to your assistant come from only your customers and can pass sensitive information to your assistant.

When configuring the JWT, you no longer need to specify the Authentication Context Class Reference (acr) claim.

1 July 2020

Salesforce support is generally available

Integrate your web chat with Salesforce so your assistant can transfer customers who asks to speak to a person to a Salesforce agent who can answer their questions. For more information, see Integrating with Salesforce.

24 June 2020

Get better answers from search skill

The search skill now has a beta feature that limits the search results that are returned to include only those for which Discovery has calculated a 20% or higher confidence score. You can toggle the feature on or off from the *Refine results to return more selective answers* switch on the configuration page. You cannot change the confidence score threshold from 0.2. This beta feature is enabled by default. For more information, see IBM Watson® Discovery search integration setup.

3 June 2020

Zendesk support is generally available

Integrate your web chat with Zendesk so your assistant can transfer customers who asks to speak to a person to a Zendesk agent who can answer their questions. And now you can secure the connection to Zendesk. For more information, see Integrating with Zendesk.

Pricing plan changes

We continue to revamp the overall service plan structure for Watson Assistant. In April, we announced a new low cost entry point for the Plus plan. Today, the Standard plan is being retired. Existing Standard plan users are not impacted; they can continue to work in their Standard instances. New users do not see the Standard plan as an option when they create a service instance. For more information, see the Pricing page.

27 May 2020

Full language support for new system entities

The new version of the system entities is generally available in dialog skills of all languages, including Arabic, Chinese (Simplified), Chinese (Traditional), Korean, and Japanese. For more information, see Supported languages.

New system entities are enabled automatically

All new dialog skills use the new version of the system entities automatically.

22 May 2020

Spelling correction in v2 API

The v2 `message` API now supports spelling correction options. For more information see the API Reference.

21 May 2020

Preview link URL change

The URL for the preview link was changed. If you previously shared the link with teammates, provide them with the new URL.

15 May 2020

Private endpoints support is available in Plus plan

You can use private endpoints to route services over the IBM Cloud private network instead of the public network. For more information, see Private network endpoints. This feature was previously available to users of Premium plans only.

14 May 2020

Get skill owner information

The email address of the person who owns the service instance that you are using is displayed from the User account menu. This information is especially helpful if you want to contact the instance owner to request access changes.

System entity deprecation

As stated in the March deprecation notice, the `@sys-location` and `@sys-person` system entities that were available as a beta feature are deprecated. If you are using one of these system entities in your dialog, a toggle is displayed for the entity on the *System entities* page. You can search your dialog

to find out where you are currently using the entity, and remove it. Consider using a contextual entity to identify references to locations and people instead. After removing the entity from your dialog, disable the entity from the *System entities* page.

13 May 2020

Stateless v2 message API

The v2 runtime API now supports a new stateless `message` method. If you have a client application that manages its own state, you can use this new method to take advantage of many of the benefits of the v2 API without the overhead of creating sessions. For more information, see the API Reference.

30 April 2020

Web chat is generally available!

Add your assistant to your company website as a web chat widget that can help your customers with common questions and tasks. Service desk transfer support continues to be a beta feature. For more information, see Web chat overview.

Secure your web chat

Enable the beta security feature of web chat so that you can verify that messages sent to your assistant come from only your customers and can pass sensitive information to your assistant.

27 April 2020

Add personality to your assistant in web chat

You can add an assistant image to the web chat header to brand the window. You can add an avatar image that represents your assistant or a brand logo, for example. For more information, see Configuring style and appearance.

Know your plan

Now your service plan is displayed in the page header. And if you have a Plus Trial plan, you can see how many days are left in the trial.

21 April 2020

Fuzzy matching support was expanded

Added support for stemming and misspelling in French, German, and Czech dialog skills. This enhancement means that the assistant can recognize an entity value that is defined in its singular form but mentioned in its plural form in user input. It also can recognize conjugated forms of a verb that is specified as an entity value.

For example, if your French-language dialog skill has an entity value of `animal`, it recognizes the plural form of the word (`animaux`) when it is mentioned in user input. If your German-language dialog skill has the root verb `haben` as an entity value, it recognizes conjugated forms of the verb (`hast`) in user input as mentions of the entity.

2 April 2020

New and improved access control

Now, when you give other people access to your Watson Assistant resources, you have more control over the level of access they have to individual skills and assistants. You can give one person read-only access to a production skill and manager-level access to a development skill, for example. For more information, see Managing access.

If you can't access the API Details for a skill or assistant anymore, you might not have the access role that is required to use the instance-level API credentials. You can use a personal API key instead.

1 April 2020

Plus plan changes

The Plus plan is now available starting at $120/month for 1,000 users on pay-as-you-go or subscription IBM Cloud accounts. And you can subscribe without contacting Sales.

French language beta support added for contextual entities

You can add contextual entities to French-language dialog skills. For more information about contextual entities, see Creating entities.

New API version

The current API version is now `2020-04-01`. The following change was made with this version:

- An `integrations` property was added to the V2 `/message` context. The service now expects the `context.integrations` property to conform to a specific schema in which the allowed values are as follows:
  - `chat`
  - `facebook`
  - `intercom`
  - `liveengage`
  - `salesforce`
  - `slack`
  - `service_desk`
  - `text_messaging`
  - `voice_telephony`
  - `zendesk`

If your app uses a `context.integrations` property that does not conform to the schema, a 400 error code will be returned.

31 March 2020

The web chat integration was updated

The update adds an `isTrackingEnabled` parameter. You can add this parameter and set it to `false` to add the `X-Watson-Learning-Opt-Out` header to each `/message` request that originates from the web chat. For more information about the header, see Data collection.

26 March 2020

The Covid-19 content catalog is available in Brazilian Portuguese, French, and Spanish

The content catalog defines a group of intents that recognize the common types of questions people ask about the novel coronavirus. You can use the catalog to jump-start development of chatbots that can answer questions about the virus and help to minimize the anxiety and misinformation associated with it. For more information about how to add a content catalog to your skill, see Using content catalogs.

19 March 2020

A Covid-19 content catalog is available

The English-only content catalog defines a group of intents that recognize the common types of questions people ask about the novel coronavirus. The World Health Organization characterized COVID-19 as a pandemic on 11 March 2020. You can use the catalog to jump-start development of chatbots that can answer questions about the virus and help to minimize the anxiety and misinformation associated with it. For more information about how to add a content catalog to your skill, see Using content catalogs.

Fixed a problem with missing User Conversation data

A recent change resulted in no logs being shown in the User Conversations page unless you had a skill as the chosen data source. And the chosen skill had to be the same skill (with same skill ID) that was connected to the assistant when the user messages were submitted.

18 March 2020

Technology preview is discontinued

The technology preview user interface was replaced with the Watson Assistant standard user interface. If you used an Actions page to create actions and steps for your skill previously, you cannot access the Actions page anymore. Instead, use the Intents and Dialog pages to work with your skill.

16 March 2020

Instructions updated for Slack integrations

The steps required to set up a Slack integration have changed to reflect permission assignment changes that were made by Slack. For more information, see Integrating with Slack.

Order of response types is preserved

Previously, if you included a response type of **Search skill** in a list of response types for a dialog node, the search results were displayed last despite its placement in the list. This behavior was changed to show the search results in the appropriate order, namely in the sequence in which the search skill response type is listed for the dialog node.

10 March 2020

Contextual entity support is generally available

You can add contextual entities to English-language dialog skills. For more information about contextual entities, see Creating entities.

French language support added for autocorrection

Autocorrection helps your assistant understand what your customers want. It corrects misspellings in the input that customers submit before the input is evaluated. With more precise input, your assistant can more easily recognize entity mentions and understand the customer's intent. For more information, see Autocorrecting user input.

The new system entities are used by new skills

For new English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills, the new system entities are enabled automatically. If you decide to turn on a system entity and add it to your dialog, it's the new and improved version of the system entity that is used. For more information, see System entities.

6 March 2020

Transfer a web chat conversation to a human agent

Delight your customers with 360-degree support by integrating your web chat with a third-party service desk solution. When a customer asks to speak to a person, you can connect them to an agent through a service desk solution, such as Zendesk or Salesforce. Service desk support is a beta feature. For more information, see Adding contact center support.

2 March 2020

Known issue accessing logs

If you cannot access user logs from the Analytics page, ask the owner of the service instance for the skill to change your service level access to make you a Manager of the instance. For more information about access control, see Managing access.

1 March 2020 deprecation notice

March 2020 deprecation notice

To help us continue to improve and expand the capabilities of the assistants you build with Watson Assistant, we are deprecating some of the older technologies. Support for the older technologies will end in June 2020. Take action now to test and adopt the new technologies, so your skills and assistants will be ready when the old technologies stop being supported.

The following technologies are being deprecated:

- **Legacy version of numeric system entities**

  We released a whole new infrastructure for our numeric system entities across all languages except Chinese, Korean, Japanese, and Arabic. The updated `@sys-number`, `@sys-date`, `@sys-time`, `@sys-currency`, and `@sys-percentage` entities provide superior number recognition with higher precision. For more information about the new system entities, see System entities.

  The old version of the numeric system entities will stop being supported in June 2020 for English, Brazilian Portuguese, Czech, Dutch, French, German, Italian, and Spanish dialog skills.

  Action: In each dialog skill where you use numeric system entities, go to the **Options>System entities** page and turn on the new system entities. Take some time to test the new version of system entities with your own dialogs to make sure they continue to work as expected. As you adopt the new system entities, share your feedback about your experience with the new technology.

- **Person and location system entities**

  The `@sys-person` and `@sys-location` system entities, which were available in English as a beta only, are being deprecated. Consider using contextual entities as a way to capture these types of proper nouns. Instead of trying to add a dictionary-based entity that covers every permutation of the names for people or cities, for example, you can teach your skill to recognize the context in which such names are used. For more information about contextual entities, see Annotation-based method.

  Action: Remove references to `@sys-person` and `@sys-location` from your dialogs. Turn off the `@sys-person` and `@sys-location` system entities to prevent yourself or others from adding them to a dialog inadvertently.

- **Irrelevance detection**

  We revised the irrelevance detection classification algorithm to make it even smarter out of the box. Now, even before you begin to teach the system about irrelevant requests, it is able to recognize user input that your skill is not designed to address. For more information, see Irrelevance detection.

  Action: In each dialog skill, go to the **Options>Irrelevance detection** page and turn on the new classification model. Make sure everything works as well, if not better, than it did before. Share your feedback.

- **Old API version dates**

  v1 API versions that are dated on or before `2017-02-03` are being deprecated. When you send calls to the service with earlier API version dates, they will receive properly formatted and valid responses for a time, so you can gracefully transition to using the later API versions. However, the confidence scores and other results that are sent in the response will reflect those generated by a more recent version of the API.

  Action: Do some testing of calls with the latest version to verify that things work as expected. Some functionality has changed over the last few years. After testing, change the version date on any API calls that you make from your applications.

28 February 2020

IBM Watson® Assistant is available in IBM Cloud Pak for Data

The service can be installed on-premises in environments where IBM Cloud Pak for Data 2.5 is installed on OpenShift or standalone. Version 2.5 has been deprecated, and documentation is no longer online; we recommend upgrading to the latest version.

26 February 2020

Slot `Save it as` field retains your edits

When you edit what gets saved for a slot by using the JSON editor to edit the value of the context variable to be something other than what is specified in the **Check for** field, your changes are kept even if someone subsequently clicks the **Save it as** field.

20 February 2020

Access control changes are coming

Notifications are displayed in the user interface for anyone with Reader and Writer level access to a service instance. The notification explains that access control is going to change soon, and that what they can do in the instance will change unless they are given Manager service access beforehand.

14 February 2020

More web chat color settings

You can now specify the color of more elements of the web chat integration. For example, you can define one color for the web chat window header. You can define a different color for the user message bubble. And another color for interactive components, such as the launcher button for the chat.

13 February 2020

Track API events

Premium plan users can now use the Activity Tracker service to track how users and applications interact with IBM Watson® Assistant in IBM Cloud®. See Using IBM Cloud Activity Tracker to audit user activity.

5 February 2020

New API version

The current API version is now `2020-02-05`. The following changes were made with this version:

- When a dialog node's response type is `connect-to-agent`, the node's `title` is used as the `topic` value. Previously, `user_label` was used.
- The `alternate_intents` property is stored as a Boolean value instead of a String.

4 February 2020

Product user interface makeover

The UI has been updated to be more intuitive, responsive, and consistent across its pages. While the look and feel of the UI elements has changed, their function has not.

Requesting early access

The button you click to request participation in the early access program has moved from the Skills page to the user account menu.

24 January 2020

New system entities are now generally available in multiple languages

The new and improved numeric system entities are now generally available in all supported languages, except Arabic, Chinese, Japanese, and Korean, where they are available as a beta feature. They are not used by your dialog skill unless you enable them from the **Options>System entities** page. For more information, see System entities.

14 January 2020

Fixed an error message that was displayed when opening an instance

An error that was displayed when you launched Watson Assistant from the IBM Cloud® dashboard has been fixed. Previously, an error message that said, `Module 'ui-router' is not available! You either misspelled the module name or forgot to load it` would sometimes be displayed.

12 December 2019

Support for private network endpoints

Users of Premium plans can create private network endpoints to connect to Watson Assistant over a private network. Connections to private network endpoints do not require public internet access. For more information, see Private network endpoints.

Full support for IBM Cloud IAM

Watson Assistant now supports the full implementation of IBM Cloud Identity and Access Management (IAM). API keys for Watson services are no longer limited to a single service instance. You can create access policies and API keys that apply to more than one service, and you can grant access between services.

- To support this change, the API service endpoints use a different domain and include the service instance ID. The pattern is `api.{location}.{offering}.watson.cloud.ibm.com/instances/{instance_id}`.

  Example URL for an instance hosted in the Dallas location: `api.us-south.assistant.watson.cloud.ibm.com/instances/6bbda3b3-d572-45e1-8c54-22d6ed9e52c2`

  The previous public endpoint domain was `watsonplatform.net`.

  For more information, see the API reference.

  These URLs do not introduce a breaking change. The new URLs work both for your existing service instances and for new instances. The original URLs continue to work on your existing service instances for at least one year (until December 2020).

- For more information, see Authenticating to Watson services.

26 November 2019

Disambiguation is available to everyone

Disambiguation is now available to users of every plan type.

The following changes were made to how it functions:

- The text that you add to the dialog **node name** field now matters.
- The text in the node name field might be shown to customers. The disambiguation feature shows it to customers if the assistant needs to ask them to clarify their meaning. The text you add as the node name must identify the purpose of the node clearly and succinctly, such as *Place*

an order or *Get plan information*.

If the *External node name* field exists and contains a summary of the node's purpose, then its summary is shown in the disambiguation list instead. Otherwise, the dialog node name content is shown.

- Disambiguation is enabled automatically for all nodes. You can disable it for the entire dialog or for individual dialog nodes.
- When testing, you might notice that the order of the options in the disambiguation list changes from one test run to the next. Don't worry; this new behavior is intended. As part of work being done to help the assistant learn automatically from user choices, the order of the options in the disambiguation list is being randomized on purpose. Changing the order helps to avoid bias that can be introduced by a percentage of people who always pick the first option without first reviewing their choices.

12 November 2019

Slot prompt JSON editor

You can now use the context or JSON editors for the slot response field where you define the question that your assistant asks to get information it needs from the customer. For more information about slots, see Gathering information with slots.

New South Korea location

You can now create Watson Assistant instances in the Seoul location. As with other locations, the IBM Cloud Seoul location uses token-based Identity and Access Management (IAM) authentication.

Technology preview

A technology preview experience was released. A select set of new users are being presented with a new user interface that takes a different approach to building an assistant.

7 November 2019

Irrelevance detection has been added

When enabled, a supplemental model is used to help identify utterances that are irrelevant and should not be answered by the dialog skill. This new model is especially beneficial for skills that have not been trained on what subjects to ignore. This feature is available for English skills only. For more information, see Irrelevance detection.

Time zone support for now() method

You can now specify the time zone for the date and time that is returned by the `now()` method. See Now().

24 October 2019

Testing improvement

You can now see the top three intents that were recognized in a test user input from the "Try it out" pane.

Error message when opening an instance

When you launch Watson Assistant from the IBM Cloud® dashboard, you might see an error message that says, `Module 'ui-router' is not available! You either misspelled the module name or forgot to load it.` You can ignore the message. Refresh the web browser page to close the notification.

14 October 2019

Deploy your assistant in minutes

Create a web chat integration to embed your assistant into a page on your website as a chat widget. See Web chat overview.

UI changes

The main menu options of **Assistants** and **Skills** have moved from being displayed in the page header to being shown as icons on the side of the page. The tabbed pages for the tools you use to develop a dialog skill were moved to a secondary navigation bar that is displayed when you open the skill.

Rich response types are supported in a dialog node with slots

You can display a list of options for a user to choose from as the prompt for a slot, for example.

Change to switching service instances

Where you go to switch service instances has changed.

Known issue: Cannot rename search skills

You currently cannot rename a search skill after you create it.

9 October 2019

New system entities changes

The following updates have been made:

- In addition to English and German, the new numeric system entities are now available in these languages: Brazilian Portuguese, Czech, French, Italian, and Spanish.
- The `part_of_day` property of the `@sys-time` entity now returns a time range instead of a single time value.

23 September 2019

Dallas updates

The updates from 20 September are now available to service instances hosted in Dallas.

20 September 2019

Inactivity timeout increase

The maximum inactivity timeout can now be extended to up to 7 days for Premium plans. See Inactivity timeout.

Pattern entity fix

A change that was introduced in the previous release which changed all alphabetic characters to lowercase at the time an entity value was added has been fixed. The case of any alphabetic characters that are part of a pattern entity value are no longer changed when the value is added.

Dialog text response syntax fix

Fixed a bug in which the format of a dialog response reverted to an earlier version of the JSON syntax. Standard text responses were being saved as `output.text` instead of `output.generic`. For more information about the `output` object, see Personalizing the dialog with context.

13 September 2019

Improved Entities and Intents page responsiveness

The Entities and Intents pages were updated to use a new JavaScript library that increases the page responsiveness. As a result, the look of some graphical user interface elements, such as buttons, changed slightly, but the function did not.

Creating contextual entities got easier

The process you use to annotate entity mentions from intent user examples was improved. You can now put the intent page into annotation mode to more easily select and label mentions. See Adding contextual entities.

6 September 2019

Label character limit increase

The limit to the number of characters allowed for a label that you define for an option response type changed from 64 characters to 2,048 characters.

12 August 2019

New dialog method

The `getMatch` method was added. You can use this method to extract a specific occurrence of a regular expression pattern that recurs in user input. For more details, see dialog methods.

9 August 2019

Introductory product tour

For some first-time users, a new introductory product tour is shown that the user can choose to follow to perform the initial steps of creating an assistant.

1 August 2019

Webhook callouts are available

Add webhooks to dialog nodes to make programmatic calls to an external application as part of the conversational flow. The new Webhook support simplifies the callout implementation process. (No more `action` JSON objects required.) For more information, see Making a programmatic call from a dialog node.

Improved dialog page responsiveness

In all service instances, the user interface of the Dialog page was updated to use a new JavaScript library that increases the page responsiveness. As a result, the look of some graphical user interface elements, such as buttons, changed slightly, but the function did not.

31 July 2019

Search skill and autocorrection are generally available

The search skill and spelling autocorrection features, which were previously available as beta features, are now generally available.

- Search skills can be created by users of Plus or Premium plans only.
- You can enable autocorrection for English-language dialog skills only. It is enabled automatically for new English-language dialog skills.

26 July 2019

Missing skills issue is resolved

In some cases, workspaces that were created through the API only were not being displayed when you opened the Watson Assistant user interface. This issue has been addressed. All workspaces that you create by using the API are displayed as dialog skills when you open the user interface.

23 July 2019

Dialog search is fixed

In some skills, the search function was not working in the Dialog page. The issue is now fixed.

17 July 2019

Disambiguation choice limit

You can now set the maximum number of options to show to users when the assistant asks them to clarify what they want to do. For more information about disambiguation, see Disambiguation.

Dialog search issue

In some skills, the search function is not working in the Dialog page. A new user interface library, which increases the page responsiveness, is being rolled out to existing service instances in phases. This search issue affects only dialog skills for which the new library is not yet enabled.

Missing skills issue

In some cases, workspaces that were created through the API only are not being displayed when you open the Watson Assistant user interface. Normally, these workspaces are displayed as dialog skills. If you do not see your skills from the UI, don't worry; they are not gone. Contact support to report the issue, so the team can enable the workspaces to be displayed properly.

15 July 2019

Numeric system entities upgrade available in Dallas

The new system entities are now also available as a beta feature for instances that are hosted in Dallas. See System entities.

12 June 2019

Numeric system entities upgrade

New system entities are available as a beta feature that you can enable in dialog skills that are written in English or German. The revised system entities offer better date and time understanding. They can recognize date and number spans, national holiday references, and classify mentions with more precision. For example, a date such as `May 15` is recognized as a date mention(`@sys-date:2019-05-15`), and is *not* also identified as a number mention (`@sys-number:15`). See System entities.

A Plus Trial plan is available

You can use the free Plus Trial plan to try out the features of the Plus plan as you make a purchasing decision. The trial lasts for 30 days. After the trial period ends, if you do not upgrade to a Plus plan, your Plus Trial instance is converted to a Lite plan instance.

23 May 2019

Updated navigation

The home page was removed, and the order of the Assistants and Skills tabs was reversed. The new tab order encourages you to start your development work by creating an assistant, and then a skill.

Disambiguation settings have moved

The toggle to enable disamibugation, which is a feature that is available to Plus and Premium plan users only, has moved. The **Settings** button was removed from the **Dialog** page. You can now enable disambiguation and configure it from the skill's **Options** tab.

An introductory tour is now available

A short product tour is now displayed when a new service instance is created. Brand new users are also given help as they start development. A new assistant is created for them automatically. Informational popups are displayed to introduce the product user interface features, and guide the new user toward taking the key first step of creating a dialog skill.

10 April 2019

Autocorrection is now available

Autocorrection is a beta feature that helps your assistant understand what your customers want. It corrects misspellings in the input that customers submit before the input is evaluated. With more precise input, your assistant can more easily recognize entity mentions and understand the customer's intent. See Autocorrecting user input for more details.

22 March 2019

Introducing search skill

A search skill helps you to make your assistant useful to customers faster. Customer inquiries that you did not anticipate and so have not built dialog logic to handle can be met with useful responses. Instead of saying it can't help, the assistant can query an external data source to find relevant information to share in its response. Over time, you can build dialog responses to answer customer queries that require follow-up questions to clarify the user's meaning or for which a short and clear response is suitable. And you can use search skill responses to address more open-ended customer queries that require a longer explanation. This beta feature is available to users of Premium and Plus service plans only.

See IBM Watson® Discovery search integration setup for more details.

4 March 2019

Simplified navigation

The sidebar navigation with separate *Build*, *Improve*, and *Deploy* tabs has been removed. Now, you can get to all the tools you need to build a dialog skill from the main skill page.

Improve page is now called Analytics

The informational metrics that Watson generates from conversations between your users and your assistant moved from the *Improve* tab of the sidebar to a new tab on the main skill page called **Analytics**.

28 February 2019

New API version

The current API version is now `2019-02-28`. The following changes were made with this version:

- The order in which conditions are evaluated in nodes with slots has changed. Previously, if you had a node with slots that allowed for digressions away, the `anything_else` root node was triggered before any of the slot level Not found conditions could be evaluated. The order of operations has been changed to address this behavior. Now, when a user digresses away from a node with slots, all the root nodes except the `anything_else` node are processed. Next, the slot level Not found conditions are evaluated. And, finally, the root level `anything_else` node is processed. To better understand the full order of operations for a node with slots, see Slot usage tips.
- Strings that begin with a number sign (#) in the `context` or `output` objects of a message are no longer treated as intent references.

  Previously, these strings were treated as intents automatically. For example, if you specified a context variable, such as `"color":"#FFFFFF"`, then the hex color code (#FFFFFF) would be treated as an intent. Your assistant would check whether an intent named #FFFFFF was detected in the user's input, and if not, would replace #FFFFFF with `false`. This replacement no longer occurs.

Similarly, if you included a number sign (#) in the text string in a node response, you used to have to escape it by preceding it with a back slash (`\`). For example, `We are the \#1 seller of lobster rolls in Maine.` You no longer need to escape the `#` symbol in a text response.

This change does not apply to node or conditional response conditions. Any strings that begin with a number sign (#) which are specified in conditions continue to be treated as intent references. Also, you can use SpEL expression syntax to force the system to treat a string in the `context` or `output` objects of a message as an intent. For example, specify the intent as `<? #intent-name ?>`.

25 February 2019

Slack integration enhancement

You can now choose the type of event that triggers your assistant in a Slack channel. Previously, when you integrated your assistant with Slack, the assistant interacted with users through a direct message channel. Now, you can configure the assistant to listen for mentions, and respond when it is mentioned in other channels. You can choose to use one or both event types as the mechanism through which your assistant interacts with users.

11 February 2019

Integrate with Intercom

Intercom, a leading customer service messaging platform, has partnered with IBM to add a new agent to the team, a virtual Watson Assistant. You can integrate your assistant with an Intercom application to enable the app to seamlessly pass user conversations between your assistant and human support agents. This integration is available to Plus and Premium plan users only. See Integrating with Intercom for more details.

8 February 2019

Version your skills

You can now capture a snapshot of the the intents, entities, dialog, and configuration settings for a skill at key points during the development process. With versions, it's safe to get creative. You can deploy new design approaches in a test environment to validate them before you apply any updates to a production deployment of your assistant. See Managing workflow with versions for more details.

Arabic content catalog

Users of Arabic-language skills can now add prebuilt intents to their dialogs. See Using content catalogs for more information.

17 January 2019

Czech language support is generally available

Support for the Czech language is no longer classified as beta; it is now generally available. See Supported languages for more information.

Language support improvements

The language understanding components were updated to improve the following features:

- German and Korean system entities
- Intent classification tokenization for Arabic, Dutch, French, Italian, Japanese, Portuguese, and Spanish

4 January 2019

IBM Cloud Functions in DC and London locations

You can now make programmatic calls to IBM Cloud Functions from the dialog of an assistant in a service instance that is hosted in the London and

Washington, DC data centers. See Making programmatic calls from a dialog node.

New methods for working with arrays

The following SpEL expression methods are available that make it easier to work with array values in your dialog:

- **JSONArray.filter**: Filters an array by comparing each value in the array to a value that can vary based on user input.
- **JSONArray.includesIntent**: Checks whether an `intents` array contains a particular intent.
- **JSONArray.indexOf**: Gets the index number of a specific value in an array.
- **JSONArray.joinToArray**: Applies formatting to values that are returned from an array.

See the array method documentation for more details.

13 December 2018

London data center

You can now create Watson Assistant service instances that are hosted in the London data center without syndication. See Data centers for more details.

Dialog node limit changes

The dialog node limit was temporarily changed from 100,000 to 500 for new Standard plan instances. This limit change was later reversed. If you created a Standard plan instance during the time frame in which the limit was in effect, your dialogs might be impacted. The limit was in effect for skills created between 10 December and 12 December 2018. The lower limits will be removed from all impacted instances in January. If you need to have the lower limit lifted before then, open a support ticket.

1 December 2018

Determine the number of dialog nodes

To determine the number of dialog nodes in a dialog skill, do one of the following things:

- From the tool, if it is not associated with an assistant already, add the dialog skill to an assistant, and then view the skill tile from the main page of the assistant. The *trained data* section lists the number of dialog nodes.
- Send a GET request to the /dialog_nodes API endpoint, and include the `include_count=true` parameter. For example:

```
curl -u "apikey:{apikey}" "https://{service-hostname}/assistant/api/v1/workspaces/{workspace_id}/dialog_nodes?version=2018-09-20&include_count=true"
```

  where {service-hostname} is the appropriate URL for your instance. For more details, see Service endpoint.

  In the response, the `total` attribute in the `pagination` object contains the number of dialog nodes.

27 November 2018

A new service plan, the Plus plan, is available

The new plan offers premium-level features at a lower price point. Unlike previous plans, the Plus plan is a user-based billing plan. It measures usage by the number of unique users that interact with your assistant over a given time period. To get the most from the plan, if you build your own client application, design your app such that it defines a unique ID for each user, and passes the user ID with each /message API call. For the built-in integrations, the session ID is used to identify user interactions with the assistant. See User-based plans explained for more information.

Artifact	Limit
Assistants	100

Contextual entities	20
Contextual entity annotations	2,000
Dialog nodes	100,000
Entities	1,000
Entity synonyms	100,000
Entity values	100,000
Intents	2,000
Intent user examples	25,000
Integrations	100
Logs	30 days
Skills	50

Plus plan limits

User-based Premium plan

The Premium plan now bases its billing on the number of active unique users. If you choose to use this plan, design any custom applications that you build to properly identify the users who generate /message API calls. For more information, see User-based plans.

Existing Premium plan service instances are not impacted by this change; they continue to use API-based billing methods. Only existing Premium plan users will see the API-based plan listed as the *Premium (API)* plan option.

See Watson Assistant service plan options for more information about all available service plans.

20 November 2018

Recommendations are discontinued

The Recommendations section on the Improve tab was removed. Recommendations was a beta feature available to Premium plan users only. It recommended actions that users could take to improve their training data. Instead of consolidating recommendations in one place, recommendations are now being made available from the parts of the tool where you make actual training data changes. For example, while adding entity synonyms, you can now opt to see a list of synonymous terms that are recommended by Watson.

9 November 2018

Major user interface revision

The Watson Assistant service has a new look and added features.

This version of the tool was evaluated by beta program participants over the past several months.

- **Skills**: What you think of as a *workspace* is now called a *skill*. A *dialog skill* is a container for the natural language processing training data and artifacts that enable your assistant to understand user questions, and respond to them.

Where are my workspaces? Any workspaces that you created previously are now listed in your service instance as skills. Click the **Skills** tab to see them.

- **Assistants**: You can now publish your skill in just two steps. Add your skill to an assistant, and then set up one or more integrations with which

to deploy your skill. The assistant adds a layer of function to your skill that enables Watson Assistant to orchestrate and manage the flow of information for you.

- **Built-in integrations**: Instead of going to the **Deploy** tab to deploy your workspace, you add your dialog skill to an assistant, and add integrations to the assistant through which the skill is made available to your users. You do not need to build a custom front-end application and manage the conversation state from one call to the next. However, you can still do so if you want to.

- **New major API version** : A V2 version of the API is available. This version provides access to methods you can use to interact with an assistant at run time. No more passing context with each API call; the session state is managed for you as part of the assistant layer.

What is presented in the tooling as a dialog skill is effectively a wrapper for a V1 workspace. There are currently no API methods for authoring skills and assistants with the V2 API. However, you can continue to use the V1 API for authoring workspaces. See API Overview for more details.

- **Switching data sources**: It is now easier to improve the model in one skill with user conversation logs from a different skill. You do not need to rely on deployment IDs, but can simply pick the name of the assistant to which a skill was added and deployed to use its data. See Improving across assistants.

- **Preview links from London instances** : If your service instance is hosted in London, then you must edit the preview link URL. The URL includes a region code for the region where the instance is hosted. Because instances in London are syndicated to Dallas, you must replace the `eu-gb` reference in the URL with `us-south` for the preview web page to render properly.

8 November 2018

Japanese data center

You can now create Watson Assistant service instances that are hosted in the Tokyo data center. See Data centers for more details.

30 October 2018

New API authentication process

The Watson Assistant service transitioned from using Cloud Foundry to using token-based Identity and Access Management (IAM) authentication in the following regions:

- Dallas (us-south)
- Frankfurt (eu-de)

For new service instances, you use IAM for authentication. You can pass either a bearer token or an API key. Tokens support authenticated requests without embedding service credentials in every call. API keys use basic authentication.

For all existing service instances, you continue to use service credentials (`{username}:{password}`) for authentication.

25 October 2018

Entity synonym recommendations are available in more languages

Synonym recommendation support was added for the French, Japanese, and Spanish languages.

26 September 2018

IBM Watson® Assistant is available in IBM® Cloud Private

IBM Watson® Assistant is available in IBM® Cloud Private

21 September 2018

New API version

The current API version is now `2018-09-20`. In this version, the `errors[].path` attribute of the error object that is returned by the API is expressed as a JSON Pointer instead of in dot notation form.

15 August 2018

Entity fuzzy matching support improvements

Fuzzy matching is fully supported for English entities, and the misspelling feature is no longer a Beta-only feature for many other languages. See Supported languages for details.

6 August 2018

Intent conflict resolution

The tool can now help you to resolve conflicts when two or more user examples in separate intents are similar to one another. Non-distinct user examples can weaken the training data and make it harder for your assistant to map user input to the appropriate intent at run time.

Disambiguation

Enable disambiguation to allow your assistant to ask the user for help when it needs to decide between two or more viable dialog nodes to process for a response. See Disambiguation for more details.

Jump-to fix

Fixed a bug in the Dialogs tool which prevented you from being able to configure a jump-to that targets the response of a node with the `anything_else` special condition.

Digression return message

You can now specify text to display when the user returns to a node after a digression. The user will have seen the prompt for the node already. You can change the message slightly to let users know they are returning to where they left off. For example, specify a response like, `Where were we? Oh, yes...` See Digressions for more details.

12 July 2018

Rich response types

You can now add rich responses that include elements such as images or buttons in addition to text, to your dialog. See Rich responses for more information.

Contextual entities (Beta)

Contextual entities are entities that you define by labeling mentions of the entity type that occur in intent user examples. These entity types teach your assistant not only terms of interest, but also the context in which terms of interest typically appear in user utterances, enabling your assistant to recognize never-seen-before entity mentions based solely on how they are referenced in user input. For example, if you annotate the intent user example, `I want a flight to Boston` by labeling `Boston` as a `@destination` entity, then your assistant can recognize `Chicago` as a `@destination` mention in a user input that says, `I want a flight to Chicago.` This feature is currently available for English only. See Adding contextual entities for more information.

When you access the tool with an Internet Explorer web browser, you cannot label entity mentions in intent user examples nor edit user example text.

New API version

The current API version is now `2018-07-10`. This version introduces the following changes:

- The content of the /message `output` object changed from being a `text` JSON object to being a `generic` array that supports multiple rich

response types, including `image`, `option`, `pause`, and `text`.

- Support for contextual entities was added.
- You can no longer add user-defined properties in `context.metadata`. However, you can add them directly to `context`.

Overview page date filter

Use the new date filters to choose the period for which data is displayed. These filters affect all data shown on the page: not just the number of conversations displayed in the graph, but also the statistics displayed along with the graph, and the lists of top intents and entities. See Controls for more information.

Pattern limit expanded

When using the **Patterns** field to define specific patterns for an entity value, the pattern (regular expression) is now limited to 512 characters.

2 July 2018

Jump-tos from conditional responses

You can now configure a conditional response to jump directly to another node. See Conditional responses for more details.

21 June 2018

Language updates for system entities

Dutch and Simplified Chinese language support are now generally available. Dutch language support includes fuzzy matching for misspellings. Traditional Chinese language support includes the availability of system entities in beta release. See Supported languages for details.

14 June 2018

Washington, DC data center opens

You can now create Watson Assistant service instances that are hosted in the Washington, DC data center. See Data centers for more details.

New API authentication process

The Watson Assistant service has a new API authentication process for service instances that are hosted in the following regions:

- Washington, DC (us-east) as of 14 June 2018
- Sydney, Australia (au-syd) as of 7 May 2018

IBM Cloud® is migrating to token-based Identity and Access Management (IAM) authentication.

For new service instances in the regions listed, you use IAM for authentication. You can pass either a bearer token or an API key. Tokens support authenticated requests without embedding service credentials in every call. API keys use basic authentication.

For all new and existing service instances in other regions, you continue to use service credentials (`{username}:{password}`) for authentication.

When you use any of the Watson SDKs, you can pass the API key and let the SDK manage the lifecycle of the tokens. For more information and examples, see Authentication in the API reference.

If you are not sure which type of authentication to use, view the Watson Assistant credentials by clicking the service instance from the Services section of the IBM Cloud Resource List.

25 May 2018

New sample workspace

The sample workspace that is provided for you to explore or to use as a starting point for your own workspace has changed. The **Car Dashboard** sample was replaced by a **Customer Service** sample. The new sample showcases how to use content catalog intents and other newer features to build a bot. It can answer common questions, such as inquiries about store hours and locations, and illustrates how to use a node with slots to schedule in-store appointments.

HTML rendering was added to Try it out

The "Try it out" pane now renders HTML formatting that is included in response text. Previously, if you included a hypertext link as an HTML anchor tag in a text response, you would see the HTML source in the "Try it out" pane during testing. It used to look like this:

```
Contact us at <a href="https://www.ibm.com">ibm.com</a>.
```

Now, the hypertext link is rendered as if on a web page. It is displayed like this:

Contact us at ibm.com.

Remember, you must use the appropriate type of syntax in your responses for the client application to which you will deploy the conversation. Only use HTML syntax if your client application can interpret it properly. Other integration channels might expect other formats.

Deployment changes

The **Test in Slack** option was removed.

11 May 2018

Information security

The documentation includes some new details about data privacy. Read more in Securing your assistant.

7 May 2018

Sydney, Australia data center opens

You can now create Watson Assistant service instances that are hosted in the Sydney, Australia data center. See IBM Cloud global data centers for more details.

4 April 2018

Search dialogs

You can now search dialog nodes for a given word or phrase.

15 March 2018

Introducing IBM Watson® Assistant

IBM Watson® Conversation has been renamed. It is now called IBM Watson® Assistant. The name change reflects the fact that Watson Assistant is expanding to provide prebuilt content and tools that help you more easily share the virtual assistants you build. Read Introducing Watson Assistant, an evolution of Watson Conversation and Virtual Agent for details.

New REST APIs and SDKs are available for Watson Assistant

The new APIs are functionally identical to the existing Conversation APIs, which continue to be supported. For more information about the Watson Assistant APIs, see the API Reference.

Dialog enhancements

The following features were added to the dialog tool:

- Simple variable name and value fields are now available that you can use to add context variables or update context variable values. You do not need to open the JSON editor unless you want to. See Defining a context variable for more details.
- Organize your dialog by using folders to group together related dialog nodes. See Organizing the dialog with folders for more details.
- Support was added for customizing how each dialog node participates in user-initiated digressions away from the designated dialog flow. See Digressions for more details.

Search intents and entities

A new search feature has been added that allows you to search intents for user examples, intent names, or descriptions, or to search entity values and synonyms.

Content catalogs

The new content catalogs contain a single category of prebuilt common intents and entities that you can add to your application. For example, most applications require a general `#greeting-type` intent that starts a dialog with the user. You can add it from the content catalog rather than building your own.

Enhanced user metrics

The Improve component has been enhanced with additional user metrics and logging statistics. For example, the Overview page includes several new, detailed graphs that summarize interactions between users and your application, the amount of traffic for a given time period, and the intents and entities that were recognized most often in user conversations.

12 March 2018

New date and time methods

Methods were added that make it easier to perform date calculations from the dialog. See Date and time calculations for more details.

16 February 2018

Dialog node tracing

When you use the "Try it out" pane to test a dialog, a location icon is displayed next to each response. You can click the icon to highlight the path that your assistant traversed through the dialog tree to arrive at the response. See Testing your dialog for details.

New API version

The current API version is now `2018-02-16`. This version introduces the following changes:

- A new `include_audit` parameter is now supported on most GET requests. This is an optional boolean parameter that specifies whether the response should include the audit properties (`created` and `updated` timestamps). The default value is `false`. (If you are using an API version earlier than `2018-02-16`, the default value is `true`.) For more information, see the API Reference.
- Responses from API calls using the new version include only properties with non-`null` values.
- The `output.nodes_visited` and `output.nodes_visited_details` properties of message responses now include nodes with the following types, which were previously omitted:
- Nodes with `type`=`response_condition`
- Nodes with `type`=`event_handler` and `event_name`=`input`

9 February 2018

Dutch system entities (Beta)

Dutch language support has been enhanced to include the availability of System entities in beta release. See Supported languages for details.

29 January 2018

REST API now supports new request parameters

- Use the `append` parameter when updating a workspace to indicate whether the new workspace data should be added to the existing data, rather than replacing it. For more information, see Update workspace.
- Use the `nodes_visited_details` parameter when sending a message to indicate whether the response should include additional diagnostic information about the nodes that were visited during processing of the message. For more information, see Send message.

23 January 2018

Unable to retrieve list of workspaces

If you see this or similar error messages when working in the tooling, it might mean that your session has expired. Log out by choosing **Log out** from the **User information** icon, and then log back in.

8 December 2017

Log data access across instances (Premium users only)

If you are a Premium user, your premium instances can optionally be configured to allow access to log data from workspaces across your different premium instances.

Copy nodes

You can now duplicate a node to make a copy of it and its children. This feature is helpful if you build a node with useful logic that you want to reuse elsewhere in your dialog. See Copying a dialog node for more information.

Capture groups in pattern entities

You can identify groups in the regular expression pattern that you define for an entity. Identifying groups is useful if you want to be able to refer to a subsection of the pattern later. For example, your entity might have a regex pattern that captures US phone numbers. If you identify the area code segment of the number pattern as a group, then you can subsequently refer to that group to access just the area code segment of a phone number. See Defining information to look for in customer input for more information.

5 December 2017

Redesigned UI for Intents and Entities

The `Intents` and `Entities` tabs have been redesigned to provide an easier, more efficient workflow when creating and editing entities and intents. See Creating intents and Defining information to look for in customer input for information about working with these tabs.

30 November 2017

Eastern Arabic numeral support

Eastern Arabic numerals are now supported in Arabic system entities.

29 November 2017

Improving understanding of user input across workspaces

You can now improve a workspace with utterances that were sent to other workspaces within your instance. For example, you might have multiple versions of production workspaces and development workspaces; you can use the same utterance data to improve any of these workspaces. See Improving across workspaces.

20 November 2017

GB18030 compliance

GB18030 is a Chinese standard that specifies an extended code page for use in the Chinese market. This code page standard is important for the software industry because the China National Information Technology Standardization Technical Committee has mandated that any software application that is released for the Chinese market after September 1, 2001, be enabled for GB18030. The service supports this encoding, and is certified GB18030-compliant.

9 November 2017

Intent examples can directly reference entities

You can now specify an entity reference directly in an intent example. That entity reference, along with all its values or synonyms, is used by the service classifier for training the intent. For more information, see Directly referencing an entity name in an intent example .

Currently, you can only directly reference closed entities that you define. You cannot directly reference pattern entities or system entities.

8 November 2017

New connector tool

You can use the new connector tool to connect your workspace to a Slack or Facebook Messenger app that you own, making it available as a chatbot that Slack or Facebook Messenger users can interact with. This tool is available only for the IBM Cloud US South region.

3 November 2017

Dialog updates

The following updates make is easier for you to build a dialog. (See Creating a dialog for details.)

- You can add a condition to a slot to make it required only if certain conditions are met. For example, you can make a slot that asks for the name of a spouse required only if a previous (required) slot that asks for marital status indicates that the user is married.
- You can now choose **Skip user input** as the next step for a node. When you choose this option, after processing the current node, your assistant jumps directly to the first child node of the current node. This option is similar to the existing *Jump to* next step option, except that it allows for more flexibility. You do not need to specify the exact node to jump to. At run time, your assistant always jumps to whichever node is the first child node, even if the child nodes are reordered or new nodes are added after the next step behavior is defined.
- You can add conditional responses for slots. For both Found and Not found responses, you can customize how your assistant responds based on whether certain conditions are met. This feature enables you to check for possible misinterpretations and correct them before saving the value provided by the user in the slot's context variable. For example, if the slot saves the user's age, and uses `@sys-number` in the *Check for* field to capture it, you can add a condition that checks for numbers over 100, and responds with something like, *Please provide a valid age in years*. See Adding conditions to Found and Not found responses for more details.
- The interface you use to add conditional responses to a node has been redesigned to make it easier to list each condition and its response. To add node-level conditional responses, click **Customize**, and then enable the **Multiple responses** option.

The **Multiple responses** toggle sets the feature on or off for the node-level response only. It does not control the ability to define conditional

responses for a slot. The slot multiple response setting is controlled separately.

- To keep the page where you edit a slot simple, you now select menu options to a.) add a condition that must be met for the slot to be processed, and b.) add conditional responses for the Found and Not found conditions for a slot. Unless you choose to add this extra functionality, the slot condition and multiple responses fields are not displayed, which declutters the page and makes it easier to use.

25 October 2017

Updates to Simplified Chinese

Language support has been enhanced for Simplified Chinese. This includes intent classification improvements using character-level word embeddings, and the availability of system entities. Note that the service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

Updates to Spanish

Improvements have been made to Spanish intent classification, for very large datasets.

11 October 2017

Updates to Korean

Language support has been enhanced for Korean. Note that the service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

3 October 2017

Pattern-defined entities (Beta)

You can now define specific patterns for an entity, using regular expressions. This can help you identify entities that follow a defined pattern, for example SKU or part numbers, phone numbers, or email addresses. See Pattern-defined entities for additional details.

- You can add either synonyms or patterns for a single entity value; you cannot add both.
- For each entity value, there can be a maximum of up to 5 patterns.
- Each pattern (regular expression) is limited to 128 characters.
- Importing or exporting via a CSV file does not currently support patterns.
- The REST API does not support direct access to patterns, but you can retrieve or modify patterns using the `/values` endpoint.

Fuzzy matching filtered by dictionary (English only)

An improved version of fuzzy matching for entities is now available, for English. This improvement prevents the capturing of some common, valid English words as fuzzy matches for a given entity. For example, fuzzy matching will not match the entity value `like` to `hike` or `bike`, which are valid English words, but will continue to match examples such as `lkie` or `oike`.

27 September 2017

Condition builder updates

The control that is displayed to help you define a condition in a dialog node has been updated. Enhancements include support for listing available context variable names after you enter the $ to begin adding a context variable.

31 August 2017

Improve section rollback

The median conversation time metric, and corresponding filters, are being temporarily removed from the Overview page of the Improve section. This removal will prevent the calculation of certain metrics from causing the median conversation time metric, and the conversations over time graph, to display inaccurate information. IBM regrets removing functionality from the tool, but is committed to ensuring that we are communicating accurate information to users.

Dialog node names

You can now assign any name to a dialog node; it does not need to be unique. And you can subsequently change the node name without impacting how the node is referenced internally. The name you specify is saved as a title attribute of the node in the workspace JSON file and the system uses a unique ID that is stored in the name attribute to reference the node.

23 August 2017

Updates to Korean, Japanese, and Italian

Language support has been enhanced for Korean, Japanese, and Italian. Note that the service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

10 August 2017

Accent normalization

In a conversational setting, users may or may not use accents while interacting with the service. As such, an update has been made to the algorithm so that accented and non-accented versions of words are treated the same for intent detection and entity recognition.

However, for some languages like Spanish, some accents can alter the meaning of the entity. Thus, for entity detection, although the original entity may implicitly have an accent, your assistant can also match the non-accented version of the same entity, but with a slightly lower confidence score.

For example, for the word `barrió`, which has an accent and corresponds to the past tense of the verb `barrer` (to sweep), your assistant can also match the word `barrio` (neighborhood), but with a slightly lower confidence.

The system will provide the highest confidence scores in entities with exact matches. For example, `barrio` will not be detected if `barrió` is in the training set; and `barrió` will not be detected if `barrio` is in the training set.

You are expected to train the system with the proper characters and accents. For example, if you are expecting `barrió` as a response, then you should put `barrió` into the training set.

Although not an accent mark, the same applies to words using, for example, the Spanish letter `ñ` versus the letter `n`, such as `uña` versus `una`. In this case the letter `ñ` is not simply an `n` with an accent; it is a unique, Spanish-specific letter.

You can enable fuzzy matching if you think your customers will not use the appropriate accents, or misspell words (including, for example, putting a `n` instead of a `ñ`), or you can explicitly include them in the training examples.

Note: Accent normalization is enabled for Portuguese, Spanish, French, and Czech.

Workspace opt-out flag

The REST API now supports an opt-out flag for workspaces. This flag indicates that workspace training data such as intents and entities are not to be used by IBM for general service improvements. For more information, see the API Reference

7 August 2017

`Next` and `last` date interpretation

The service treats `last` and `next` dates as referring to the most immediate last or next day referenced, which may be in either the same or a previous week. See the system entities topic for additional information.

3 August 2017

Fuzzy matching for additional languages (Beta)

Fuzzy matching for entities is now available for additional languages, as noted in the Supported languages topic.

Partial match (Beta - English only)

Fuzzy matching will now automatically suggest substring-based synonyms present in user-defined entities, and assign a lower confidence score as compared to the exact entity match. See Fuzzy matching for details.

28 July 2017

Updates

This release includes the following updates:

- When you set bidirectional preferences for the tooling, you can now specify the graphical user interface direction.
- The color scheme of the tooling was updated to be consistent with other Watson services and products.

19 July 2017

REST API now supports access to dialog nodes

The REST API now supports access to dialog nodes. For more information, see the API Reference.

14 July 2017

Slots enhancement

The slots functionality of dialogs was enhanced. For example, a *slot_in_focus* property was added that you can use to define a condition that applies to a single slot only. See Gathering information with slots for details.

12 July 2017

Support for Czech

Czech language support has been introduced. See Supported languages for additional details.

11 July 2017

Test in Slack

You can use the new **Test in Slack** tool to quickly deploy your workspace as a Slack bot user for testing purposes. This tool is available only for the IBM Cloud US South region.

Updates to Arabic

Arabic language support has been enhanced to include absolute scoring per intent, and the ability to mark intents as irrelevant. See Supported languages for additional details. Note that the service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

23 June 2017

Updates to Korean

Korean language support has been enhanced. See Supported languages for additional details. The service learning models may have been updated as part of this enhancement, and when you retrain your model any changes will be applied.

22 June 2017

Introducing slots

It is now easier to collect multiple pieces of information from a user in a single node by adding slots. Previously, you had to create several dialog nodes to cover all the possible combinations of ways that users might provide the information. With slots, you can configure a single node that saves any information that the user provides, and prompts for any required details that the user does not. See Gathering information with slots for more details.

Simplified dialog tree

The dialog tree has been redesigned to improve its usability. The tree view is more compact so it is easier to see where you are within it. And the links between nodes are represented in a way that makes it easier to understand the relationships between the nodes.

21 June 2017

Arabic support

Language support for Arabic is now generally available. For details, Supported languages.

Language updates

The service algorithms have been updated to improve overall language support. See Supported languages for details.

16 June 2017

Recommendations (Beta - Premium users only)

The Improve panel also includes a **Recommendations** page that recommends ways to improve your system by analyzing the conversations that users have with your chatbot, and taking into account your system's current training data and response certainty.

14 June 2017

Fuzzy matching for additional languages (Beta)

Fuzzy matching for entities is now available for additional languages. For more information, see Supported languages. You can turn on fuzzy matching per entity to improve the ability of your assistant to recognize terms in user input with syntax that is similar to the entity, without requiring an exact match. The feature is able to map user input to the appropriate corresponding entity despite the presence of misspellings or slight syntactical differences. For example, if you define giraffe as a synonym for an animal entity, and the user input contains the terms `giraffes` or `girafe`, the fuzzy match is able to map the term to the animal entity correctly. See Fuzzy matching for details.

13 June 2017

User conversations

The Improve panel now includes a **User conversations** page, which provides a list of user interactions with your chatbot that can be filtered by

keyword, intent, entity, or number of days. You can open individual conversations to correct intents, or to add entity values or synonyms.

Regex change

The regular expressions that are supported by SpEL functions like find, matches, extract, replaceFirst, replaceAll and split have changed. A group of regular expression constructs are no longer allowed, including look-ahead, look-behind, possessive repetition and backreference constructs. This change was necessary to avoid a security exposure in the original regular expression library.

12 June 2017

Updates

This release includes the following updates:

- The maximum number of workspaces that you can create with the **Lite** plan (formerly named the Free plan) changed from 3 to 5.
- You can now assign any name to a dialog node; it does not need to be unique. And you can subsequently change the node name without impacting how the node is referenced internally. The name you specify is treated as an alias and the system uses its own internal identifier to reference the node.
- You can no longer change the language of a workspace after you create it by editing the workspace details. If you need to change the language, you can export the workspace as a JSON file, update the language property, and then import the JSON file as a new workspace.

6 June 2017

Learn

A new *Learn about* page is available that provides getting started information and links to service documentation and other useful resources. To open the page, click the icon in the page header.

Bulk export and delete

You can now simultaneously export a number of intents or entities to a CSV file, so you can then import and reuse them for another application. You can also simultaneously select a number of entities or intents for deletion in bulk.

Updates to Korean

Korean tokenizers have been updated to address informal language support. IBM continues to work on improvements to entity recognition and classification.

Emoji support

Emojis added to intent examples, or as entity values, will now be correctly classified/extracted.

Only emojis that are included in your training data will be correctly and consistently identified; emoji support may not correctly classify similar emojis with different color tones or other variations.

Entity stemming (Beta - English only)

The fuzzy matching beta feature recognizes entities and matches them based on the stem form of the entity value. For example, this feature correctly recognizes `bananas` as being similar to `banana`, and `run` being similar to `running` as they share a common stem form. For more information, see Fuzzy matching.

Workspace import progress

When you import a workspace from a JSON file, a tile for the workspace is displayed immediately, in which information about the progress of the import is displayed.

Reduced training time

Multiple models are now trained in parallel, which noticeably reduces the training time for large workspaces.

26 May 2017

New API version

The current API version is now `2017-05-26`. This version introduces the following changes:

- The schema of ErrorResponse objects has changed. This change affects all endpoints and methods. For more information, see the API Reference.
- The internal schema used to represent dialog nodes in exported workspace JSON has changed. If you use the `2017-05-26` API to import a workspace that was exported using an earlier version, some dialog nodes might not import correctly. For best results, always import a workspace using the same version that was used to export it.

25 May 2017

Manage context variables

You can now manage context variables in the "Try it out" pane. Click the **Manage context** link to open a new pane where you can set and check the values of context variables as you test the dialog. See Testing your dialog for more information.

16 May 2017

Updates

This release includes the following updates:

- A **Car Dashboard** sample workspace is now available when you open the tool. To use the sample as a starting point for your own workspace, edit the workspace. If you want to use it for multiple workspaces, then duplicate it instead. The sample workspace does not count toward your subscription workspace total unless you use it.
- It is now easier to navigate the tool. The navigation menu options are available from the side of the main page instead of the top. In the page header, breadcrumb links display that show you where you are. You can now switch between service instances from the Workspaces page. To get there quickly, click **Back to workspaces** from the navigation menu. If you have multiple service instances, the name of the current instance is displayed. You can click the **Change** link beside it to choose another instance.
- When you create a dialog, two nodes are now added to it for you: 1) a **Welcome** node at the start the dialog tree that contains the greeting to display to the user and 2) an **Anything else** node at the end of the tree that catches any user inquiries that are not recognized by other nodes in the dialog and responds to them. See Creating a dialog for more details.
- When you are testing a dialog in the "Try it out" pane, you can now find and resubmit a recent test utterance by pressing the Up key to cycle through your previous inputs.
- Experimental Korean language support for 5 system entities (`@sys-date` , `@sys-time` , `@sys-currency` , `@sys-number` , `@sys-percentage`) is now available. There are known issues for some of the numeric entities, and limited support for informal language input.
- An Overview page is available from the Improve tab. The page provides a summary of interactions with your bot. You can view the amount of traffic for a given time period, as well as the intents and entities that were recognized most often in user conversations. For additional information, see Dialog skill analytics overview.

27 April 2017

System entities

The following system entities are now available as beta features in English only:

- sys-location: Recognizes references to locations, such as towns, cities, and countries, in user utterances.
- sys-person: Recognizes references to people's names, first and last, in user utterances.

For more information, see the System entities reference.

Fuzzy matching for entities

Fuzzy matching for entities is a beta feature that is now available in English. You can turn on fuzzy matching per entity to improve the ability of your

assistant to recognize terms in user input with syntax that is similar to the entity, without requiring an exact match. The feature is able to map user input to the appropriate corresponding entity despite the presence of misspellings or slight syntactical differences. For examples, if you define **giraffe** as a synonym for an animal entity, and the user input contains the terms `giraffes` or `girafe`, the fuzzy match is able to map the term to the animal entity correctly. See How fuzzy matching works.

18 April 2017

Updates

This release includes the following updates:

- The REST API now supports access to the following resources:
  - entities
  - entity values
  - entity value synonyms
  - logs

  For more information, see the API Reference.
- The behavior of the /messages `POST` method has changed the handling of entities and intents specified as part of the message input:
  - If you specify intents on input, your assistant uses the intents you specify, but uses natural language processing to detect entities in the user input.
  - If you specify entities on input, your assistant uses the entities you specify, but uses natural language processing to detect intents in the user input.

  The behavior has not changed for messages that specify both intents and entities, or for messages that specify neither.
- The option to mark user input as irrelevant is now available for all supported languages. This is a beta feature.
- A new Credentials tab provides a single place where you can find all of the information you need for connecting your application to a workspace, as well as other deployment options. To access the Credentials tab for your workspace, click the icon and select **Credentials**.

9 March 2017

REST API updates

The REST API now supports access to the following resources:

- workspaces
- intents
- examples
- counterexamples

For more information, see the API Reference.

7 March 2017

Intent name restrictions

The use of `.` or `..` as an intent name causes problems and is no longer supported. You cannot rename or delete an intent with this name; to change the name, export your intents to a file, rename the intent in the file, and import the updated file into your workspace. Paying customers can contact support for a database change.

1 March 2017

System entities are now enabled in German

System entities are now enabled in German.

22 February 2017

Messages are now limited to 2,048 characters

Messages are now limited to 2,048 characters.

3 February 2017

Updates

This release includes the following updates:

- We changed how intents are scored and added the ability to mark input as irrelevant to your application.
- This release introduced a major change to the workspace. To benefit from the changes, you must manually upgrade your workspace.
- The processing of **Jump to** actions changed to prevent loops that can occur under certain conditions. Previously, if you jumped to the condition of a node and neither that node nor any of its peer nodes had a condition that was evaluated as true, the system would jump to the root-level node and look for a node whose condition matched the input. In some situations this processing created a loop, which prevented the dialog from progressing.

Under the new process, if neither the target node nor its peers is evaluated as true, the dialog turn is ended. To reimplement the old model, add a final peer node with a condition of `true`. In the response, use a **Jump to** action that targets the condition of the first node at the root level of your dialog tree.

11 January 2017

Customize node titles

In this release, you can customize node titles in dialog.

22 December 2016

Node title section

In this release, dialog nodes display a new section for `node title`. The ability to customize the `node title` is not available. When collapsed, the `node title` displays the `node condition` of the dialog node. If there is not a `node condition`, "Untitled Node" is displayed as the title.

19 December 2016

Dialog editor UI changes

Several changes make the dialog editor easier and more intuitive to use:

- A larger editing view makes it easier to view all the details of a node as you work on it.
- A node can contain multiple responses, each triggered by a separate condition. For more information see Responses.

5 December 2016

Updates

This release includes the following updates:

- New languages are supported, all in Experimental mode: German, Traditional Chinese, Simplified Chinese, and Dutch.
- Two new system entities are available: @sys-date and @sys-time. For details, see System entities.

21 October 2016

Updates

This release includes the following updates:

- The service now provides system entities, which are common entities that can be used across any use case.
- You can now view a history of conversations with users on the Improve page. You can use this to understand the behavior of your chatbot.
- You can now import entities from a comma-separated value (CSV) file, which helps with when you have a large number of entities.

20 September 2016

New version 2016-09-20

To take advantage of the changes in a new version, change the value of the `version` parameter to the new date. If you're not ready to update to this version, don't change your version date.

- version **2016-09-20**: `dialog_stack` changed from an array of strings to an array of JSON objects.

29 August 2016

Updates

This release includes the following updates:

- You can move dialog nodes from one branch to another, as siblings or peers.
- You can expand the JSON editor window.
- You can view chat logs of your bot's conversations to help you understand it's behavior. You can filter by intents, entities, date, and time.

11 July 2016

General Availability

This General Availability release enables you to work with entities and dialogs to create a fully functioning bot.

18 May 2016

Experimental release

This Experimental release introduces the user interface and enables you to work with workspaces, intents, and examples.

Web chat release notes

Find out what's new in the web chat integration.

The web chat changelog lists changes ordered by version number. For more information about the web chat, see Integrating the web chat with your website.

For information about new features and improvements to the core watsonx Assistant product, see Release notes.

Controlling the web chat version

If you want to evaluate changes that are introduced in a web chat release before you apply them to your deployment, you can set a version of your web chat. For more information, see Controlling the web chat version.

8.12.0

Release date: 24 September 2025

- Fixed an issue where error messages could appear multiple times during communication failures with Assistant.
- Fixed an issue where the retry button did not function after web chat failed to open due to communication problems, even when the connection was restored.

8.11.0

Release date: 7 July 2025

- Added the ability to use the `servers` configuration option on Cloud Pak for Data.
- Made some performance enhancements to how the web chat JavaScript is loaded.
- Fixed a bug that can cause a request to be resubmitted when the chat is reloaded if the response was empty.

8.10.0

Release date: 26 June 2025

- Performance and accessibility updates.

8.9.0

Release date: 18 June 2025

- Increased the response timeout limit from 40 seconds to 120 seconds to support more complex or time-consuming operations in custom extensions and conversational skills.
- Minor bug fixes.

8.8.0

Release date: 9 May 2025

- Added a new `updateLauncherConfig` instance method.
- Added a new `allow_return` property on `updateHomeScreenConfig`.
- Minor bug fixes.

8.7.1

Release date: 20 March 2025

- Fixed a bug with the date picker closing when a user clicks on it.
- Fixed a bug that occurs when using a `pre:receive` event to change a context property from a single value to an array.

- Fixed an accessibility bug with disambiguation responses.
- Updated the launcher icon used when using the chat with the IBM AI theme.
- Fixed a bug with the ZenDesk agent app zip file that would prevent the agent app from loading in ZenDesk.

8.7.0

Release date: 17 February 2025

- Bug fixes.

8.6.0

Release date: 3 February 2025

- **Stop Streaming:** Web chat has added a "Stop Response" button to stop streamed responses that can be canceled.
  - When the user clicks the "Stop response" button, it fires a stopStreaming event that you can hook into.
- Bug fixes.

8.5.3

Release date: 21 January 2025

- **Changes to conversational search:** Made some UI changes to conversational search responses. This includes the removal of the "Accuracy of generated answers may vary." disclaimer.
- Bug fixes.

8.5.2

Release date: 6 January 2025

- Bug fixes.

8.5.1

Release date: 5 December 2024

- Enhanced table response type to allow users to change the number of rows visible per page.
- Fixed bug that caused the typing indicator to not be visible in browsers other than Chrome.

7.11.0

Release date: 21 November 2024

- Added an `asyncCallout` option to the `instance.send` method that can be used to ensure a call to an extension is done using a one step process instead of the default two step process.

8.5.0

Release date: 18 November 2024

- Fixed a minor accessibility issue.

8.4.0

Release date: 21 October 2024

- **Table response type:** Beta support for the new table response type is now available in web chat. For more information, see Response types reference.
- Fixed styling issues for single card carousels.
- Fixed a bug that would sometimes cause an error message to be shown when the user clicks a link.

8.3.3

Release date: 14 October 2024

- **Accessibility fixes:** Added elements to web chat to make it easier to navigate the chat using the next and previous heading keys while using a screen reader.
- Fixed a focus issue with the custom menu. The menu will remain focused after the user selects an item from within the menu.
- Extensions can no longer cause the input field to become disabled while waiting for the extension to complete.

8.3.2

Release date: 23 September 2024

- Fixed a bug that broke the Journeys feature.

8.3.1

Release date: 13 September 2024

- Bug fixes.

8.3.0

Release date: 9 September 2024

- **Rich text support in conversational search:** Conversational search responses now support rich text formatting in responses, including Markdown.
- **Changes to conversational search sources:** Sources are no longer highlighted in conversational search responses by default. To see the highlights, you must expand the sources carousel and select the source.
- Added the region `aws-eu-central-1`.
- Fixed a bug in the NiceDFO service desk integration that made custom fields unavailable in some cases.
- Fixed a bug with the `aiTooltipAfterDescriptionElement` writeable element that prevented the content from being visible for some occurrences of the tooltip.
- Fixed a bug with the `instance.send` method when called from a `restartConversation` event that would cause web chat to incorrectly display the greeting message.

8.2.6

Release date: 20 August 2024

- Minor improvements.

8.2.5

Release date: 12 August 2024

- **Updated behavior of a search result:** You can now click the web chat card to see more information about the search result if the result is not contained in a URL or PDF file.
- Updated styling for errors received by the web chat.

8.2.4

Release date: 5 August 2024

- Fixed a bug that could cause the web chat to stop working if streaming from an extension is used.

8.2.3

Release date: 29 July 2024

- **Updated markdown behavior:** Fixed a problem with numbers in ordered lists not showing the right numbers.
- **Improved message responsiveness:** Removed or reduced the delay added to longer responses.
- Bug fixes.

8.2.2

Release date: 15 July 2024

- **Updated visuals for search results:** The visuals for search responses are updated to make it more consistent with conversational search. In addition, the web chat now displays the search results in a carousel.
- **Updates to dropdown fields:** The maximum width for the dropdown fields for `option` responses are removed. The width of the dropdown fields is now set to 100% of the width of the chat window. However, if you use the `hasContentMaxWidth: true` option, the width is limited to 380px.
- Bug fixes.

8.2.1

Release date: 1 July 2024

- Reverted the changes related to the line breaks and bullets in your assistant's responses, which was introduced per improvements in markdown rendering in the previous release.

8.2.0

Release date: 20 June 2024

- **New layout configuration:** You can customize a full-screen chat window by setting a max-width and removing the drop shadow that provides seamless integration for your assistant. For more information, see Layout configuration.
- **New instance method:** You can change the title in the header of the chat with the new `updateMainHeaderTitle`. For more information, see instance.updateMainHeaderTitle.
- **New Theme config option:** You can change the type of corners to give the web chat widget to be either "rounded" or "squared". For more information, see Theming configuration.
- **New Home Screen config option:** You can hide the **Home Screen** avatar, greeting message, and starters using the `homeScreenConfig.custom_content_only` so only custom content is visible. For more information, see instance.updateHomeScreenConfig.
- Improved markdown handling of tables, code blocks, block quotes, and fenced code blocks.
- You can now see a change in line breaks and bullets in your assistant's responses due to the improvements made in rendering markdown in the web chat.
- Bug fixes

8.1.3

Release date: 10 June 2024

- Minor bug fixes.

8.1.2

Release date: 3 June 2024

- Minor bug fixes.

8.1.1

Release date: 10 May 2024

- Minor bug fixes.

8.1.0

Release date: 15 April 2024

- **Support for conversational search streaming**: You can now enable or disable the conversational search streaming in web chat. To enable the conversational search streaming, go to **Integrations** > **Web chat** > **Style** in your assistant. Currently, you can enable streaming only for conversational search responses. However, your customers can see the search message stream in real time in their web chat.
- **Accent colors for home screen** : You can now apply accent colors instead of the default background colors on the web chat home screen. You can

also control the gradient that is displayed on the default home screen background. To change the background color on the web chat home screen, go to **Integrations** > **Web chat** > **Home screen** in your assistant.

- **Updates to `restartConversation()` method**: You can now use the `restartConversation()` method to delete the watsonx Assistant session from the server in addition to deleting the watsonx Assistant session from the client.
- **New `sessionExpired` event**: Your assistant now triggers a new `sessionExpired` event before the assistant session expires. For more information, see sessionExpired event.

8.0.1

Release date: 2 April 2024

- Fixed a bug that would prevent the web chat from loading if `clientVersion` was set to `'latest'`, `'8'`, or `'8.0'`. This bug did not affect web chats locked on previous versions.

8.0.0

Release date: 1 April 2024

- **New Carbon UI for web chat**: The web chat has an upgraded Carbon UI for AI, Carbon 11, that helps in AI explainability and adding visual cues to make your interaction AI responsive.

  You must update the Carbon design before you upgrade to web chat 8.0.0 because the new Carbon UI conflicts with the user defined responses and writeable elements.

  In addition, the new Carbon 11 design system does not support Carbon 10 components. Therefore, addition of the Carbon 10 components in to a web chat with the Carbon 11 design system might disrupt the design styling. However, you can add the Carbon 11 components in to an existing web chat with the Carbon 10 design system because the Carbon 11 components work along with the Carbon 10 components in the same code base without any design disruption. Therefore, you can upgrade any individual Carbon 10 component to Carbon 11 component in an existing web chat without disrupting the remaining Carbon 10 components.

  The new updated home screen design is no longer a solid accent color, although a future update will introduce control to the colors.

- **New theme configuration option**: You can now use a new `themeConfig` property object in web chat that replaces the `carbonTheme` configuration property. The following new options are available in the configuration object:

  - `carbonTheme`

    Select a Carbon theme for your widget per the Carbon Design System.

  - `useAITheme`

    To add IBM Carbon for AI branding to the web chat.

  For more information, see Web chat theme configuration.

- **Updated message API version**: Web chat now uses the `2023-06-15` version of the watsonx Assistant API. The previous version was `2021-09-08`. For more information about changes in behavior for watsonx Assistant with the new API version, see Release notes for watsonx Assistant .

- **New CSP requirements**: You must use the new CSP requirements for web chat. For more information, see Web chat architecture security.

- **Changes in custom service desk** : All the custom service desks must now implement a `getName()` function. If you are using a custom service desk without the `getName()` function, you must update your service desks before you upgrade to Web chat `8.0.0`. For more information, see Custom Service Desk API details.

- **Changes in page direction setting**: The `direction` configuration option is not available in web chat now because web chat now uses the `dir` attribute in the `html` element. In addition, the direction configuration is set to left-to-right by default.

- **Native Carbon classes replace the home screen help classes** : The native Carbon classes now replace the home screen help classes in web chat. For more information, see Web chat CSS helper classes.

- **The loadWatsonAssistant.js file is no longer updated** : Starting from version 8.0.0, the `loadWatsonAssistantChat.js` file, which was used by the embed script to load the web chat, is no longer updated. Therefore, for web chat version 8.0.0, you must use the `WatsonAssistantChatEntry.js` file in the embed script to load the web chat instead of the `loadWatsonAssistantChat.js` file. For web chat versions `7.x` and earlier, you can continue to use the `loadWatsonAssistantChat.js` file in the embed script even though it is replaced with the `WatsonAssistantChatEntry.js` file. To make any changes to the embed script, you access the full embed script in the `Embed` tab in the web chat settings.

- **Deprecated window methods**: In web chat version 8.0.0, the following window methods are deprecated: - `openWindow` - `closeWindow` - `toggleOpen`

$ However, watsonx Assistant continues to support these window methods in web chat version 8.0.0. For more information about migrating from window to view methods, see [Migrating from window to view events and methods](https://web-chat.global.assistant.watson.cloud.ibm.com/docs.html?to=api-events#windowviewmigration){: external}.

- **Deprecated events**: In web chat version 8.0.0, the following events are deprecated: - `window:pre:open` - `window:open` - `window:pre:close` - `window:close`

  $ However, watsonx Assistant continues to support these events in web chat version 8.0.0. For more information about migrating from window to view events, see [Migrating from window to view events and methods](https://web-chat.global.assistant.watson.cloud.ibm.com/docs.html?to=api-events#windowviewmigration){: external}.

7.10.0

Release date: 31 January 2024

- **Show timestamps on messages**: You can now display timestamps on top of messages in the web chat with the `enableMessageTimestamps` configuration option. For more information, see web chat configuration.
- **New response types**: You now have the option in web chat to render the new card, grid, carousel and button response types. For more information, see Response types reference.

7.9.0

Release date: 11 December 2023

- **Enhanced sources view for conversational search** : In the conversational search feature of web chat, you can now access *sources* of the highlighted response texts in a collapsible carousel. In addition, you can select and find the source for each highlighted text in a response by clicking on it. For more information, see Conversational search.
- **Added error handling to the built-in PDF viewer** : If any error occurs while opening a PDF document in its viewer, the web chat now automatically opens the document in a new tab of the web browser by using the native PDF viewer. This acts as a workaround for the issues such as unsupported CORS (Cross-origin resource sharing). However, in some browsers, the web chat generates a `popup-blocked` error while opening the document.
- **Added an option to disable the built-in PDF viewer** : You can now disable the PDF viewer in web chat by configuring `disablePDFViewer`. When the PDF viewer is `disabled`, the documents open in a new tab by using the native PDF viewer in the web browser. For more information, see web chat configuration.
- **New property in web chat state object** : In the object returned from the getState() instance method, the new `userID` property contains the current ID of the web chat. For example, by using this feature, you can get information about the anonymous user ID that web chat assigns to a user by default if no user ID is created.

7.8.0

Release date: 13 November 2023

- **Added a built-in PDF viewer**: A built-in viewer for PDF files can be used for search results from watsonx Discovery that contains links to the PDF documents. The document links must support CORS (Cross-origin resource sharing) to open in web chat.
- **Enhanced routing configuration for the Salesforce integration**: The `additional_routing_info` data passed from your assistant to web chat now has `button_id` and `button_overrides` properties that can further control how web chat routes the user to an agent. The `button_ids` property is deprecated, and `button_overrides` is used instead. And all properties in the `additional_routing_info` objects are optional. If the value for a property is not included, web chat defaults to the value in live agent settings from the watsonx Assistant web chat configuration. For more information, see Routing information.
- **Modified the destroySession method**: The destroySession instance method was enhanced to delete the session from the watsonx Assistant servers and remove session information from the browser memory when called.

7.7.1

Release date: 16 October 2023

- Bug fixes.

7.7.0

Release date: 5 October 2023

- **Added `fullWidth` to custom responses**: The `customResponse` event includes a `fullWidth` property that can be set to indicate to web chat of a custom response to be rendered full width in the main window. For more information, see customResponse event.
- **Longer message input is allowed**: The maximum number of characters in the message input field is increased from 300 to 2048.

- **File uploads to Salesforce**: The Salesforce integration supports the ability for users to upload files to an agent when the agent requests a file from the user. If your website has a content security policy, you might need to add `*.salesforce.com` to allow web chat to connect to the Salesforce endpoint used for uploading files.
- **Reconnecting to Salesforce agents**: The Salesforce integration automatically reconnects the user to the agent if web chat reloads while a user is engaged in conversation with an agent.
- **New branding**: The IBM watermark reflects the new IBM watsonx brand.

7.6.0

Release date: 21 August 2023

- **Added transaction ID to `onError` function:** Information in the `onError` config function contains the transaction ID in the `transactionID` property when some errors communicating with watsonx Assistant occur. For more information, see Listening for errors.
- Fixed a bug that occurred when sending two messages to an extension without waiting.
- Fixed a bug where the web chat launcher did not render when canceling the `window:open` event during session history.
- Bug fixes.

7.5.0

Release date: 31 July 2023

- **New viewing options for the Journeys beta feature** : With Journeys adding a third view to web chat -- the other two views being the launcher and main window -- we moved away from methods and events that focused on opening and closing the main window. These events were deprecated in favor of a more generic view change system that includes a new instance.changeView method, and view:pre:change and view:change events. You have more flexibility and can open multiple views at the same time.

  Window methods and events are supported for existing assistants. If you add tours to your assistant, or are using tours, your window events and methods might not work as expected and might not be supported. If you wish to add tours to your assistant, you need to update your custom code to use the new view change methods and events.

  The current window methods and events are still supported for existing web chats and assistants but are deprecated and will be removed in a future version. For more information, see Migrating from window to view events and methods.
- **New Genesys Web Messenger service desk integration**: The service desk integration for Genesys Web Messenger is no longer in beta, and now includes support for user information strings in more languages. For more information, see Integrating with Genesys Web Messenger.
- **New NICE CXone service desk integration** : The service desk integration for NICE CXone Digital First Omnichannel is released. For more information, see Integrating with NICE CXone Digital First Omnichannel.
- **Added a restart button**: A new `showRestartButton` configuration option specifies whether the web chat interface needs to display a restart button in the header, in addition to the existing - (Minimize) button. A customer can click this button to end the conversation or end any conversation with a live agent, while keeping the chat open. The chat transcript is cleared, but any transcript of a conversation with a live agent is preserved. For more information, see showRestartButton.
- **Carbon charts**: Carbon charts are supported in custom responses.
- **Reconnecting with a custom service desk integration** : Web chat now provides support to custom service desk integrations to allow them to reconnect the user to an agent when web chat is reloaded. For more information, see Reconnecting sessions.
- **Screen sharing with a custom service desk integration** : Web chat provides support to custom service desk integrations to allow and to give the user options to agree to or stop a screen-sharing session with a service desk. The actual screen sharing capability is not provided by web chat; it must be provided by the service desk integration. Screen sharing.
- The `history` property in message objects has a newly added `from_history` property that indicates whether the message came from session history. For more information, see Message object extensions.

7.4.0

Release date: 12 June 2023

- **Added CSS variables for customizing the launcher**: For more information, see instance.updateCSSVariables.

7.3.0

Release date: 30 May 2023

- **Released a beta of Genesys Web Messenger service desk integration** , which currently includes user information strings in English. For more information, see Genesys Web Messenger.
- **Added beta support for file sharing with custom service desk integrations** that currently includes user information strings in English. For more information, see Custom service desks.

7.2.2

Release date: 1 May 2023

- Bug fixes.

7.2.1

Release date: 24 April 2023

- Bug fixes.

7.2.0

Release date: 10 April 2023

- Added support for inline iframe responses. For more information, see Response types reference.
- Redesigned the agent conversation experience.
- Bug fixes.

7.1.1

Release date: 13 February 2023

- **New journey events**: The new tour:start, tour:end, and tour:step events provide details about the user's progress through a journey (also known as a *tour*). These events can be used to navigate to a specific page when the user starts a journey or reaches a certain step, or to show a survey after a journey ends.
- **New journey instance methods**: The new tours object supports instance methods that provide better control over journeys. You can use these methods to start or end a journey, or to automatically navigate through a journey in response to user actions.
- **Added journey strings to the language pack** : New strings for journeys were added to the language pack. You can modify the strings in the language pack by using the updateLanguagePack() instance method. For more information about language packs, see Languages.

For more information about the journeys beta feature, see Guiding customers with journeys.

7.1.0

Release date: 17 January 2023

- **Updated Zendesk agent app** : The agent app for Zendesk is compatible with Zendesk workspaces.
- In the service desk starter kits, the instance of the web chat integration was added to the serviceDeskFactory parameters to make it accessible to custom service desk implementations.
- **New instance methods**: The new elements.getMessageInput() and elements.getHomeScreenInput() instance methods enable access to the input fields used by the customer to send messages. You can use these methods to change the input or to execute an action as the user is typing (for example, to implement for a type-ahead feature).
- **New event**: The new agent:pre:sessionHistory event filters potential PII from messages sent from a customer or service desk agent before the messages are sent to the assistant for storage in the session history.
- **New property in web chat state object** : In the object returned from the getState() instance method, the new `isDebugEnabled` property indicates whether the web chat debug flag is set to `true`.

7.0.0

Release date: 5 December 2022

- **Streamlined live agent handoff**: The live agent handoff experience is streamlined and simplified. Instead of opening the live agent chat in a separate

window, the web chat now shows the live agent entering the conversation in the same window.

Because of this change, the updateCustomMenuOptions instance method reflects a single view, with a single list of custom menu options. If you want to customize menu options only during a live agent chat, you can subscribe to the agent:pre:startChat and agent:endChat events to trigger your customizations.

- `agent:endChat` **changes**: The agent:endChat event now also fires if the customer cancels a live agent request before the agent joins. If you want to show a post-chat form only after a live agent chat, you can use the new `requestCancelled` flag on the event to determine whether the request was canceled.
- **New configuration options**: The following new options are available in the configuration object:
  - `serviceDesk.availabilityTimeoutSeconds`: Specifies how long the web chat waits for an available agent before automatically canceling the live agent request.
  - `serviceDesk.disableAgentSessionHistory`: Disables storage of live agent chats in the session history. If this option is set to `true`, live agent chat history is not stored; this means that if the web chat is reloaded, the live agent chat history is lost.

  For more information, see Service desk options.
- `elements` **instance property**: A new elements instance property provides methods that you can use to apply CSS styles to individual elements used by the web chat. (Only the main window is supported.)
- `skip_card` **option for journeys**: Support added for a new `skip_card` property in the journeys beta feature. You can use this property to start a journey immediately without waiting for the customer to click the introductory card, or even to start a journey from your website without opening the web chat at all. For more information, see Guiding customers with journeys.
- **Path changes**: Some internal paths for communication with the assistant were changed. If you have firewall or proxy rules that are configured to allow specific paths, you might need to update your configuration to allow the following paths:
  - `/<SUBSCRIPTION_ID>/chat/<INTEGRATION_ID>/config`
  - `/<SUBSCRIPTION_ID>/chat/<INTEGRATION_ID>/message`

6.9.0

Release data: 14 November 2022

- You can now create *journeys* to guide your customers through tasks they can already complete on your website. A journey is an interactive, multipart response that can combine text, video, and images, presented in sequence in a small window superimposed over your website.

  Journeys are available as a beta feature. For more information, see Guiding customers with journeys.

6.8.1

Release date: 7 November 2022

- Bug fixes.

6.8.0

Release date: 31 October 2022

- Added support for sending pre-chat information to the Salesforce integration.

6.7.0

Release date: 10 October 2022

- **New** `updateIsTypingCounter()` **method**: The new `updateIsTypingCounter()` instance method updates the counter that determines whether the typing indicator is displayed. For more information, see instance.updateIsTypingCounter().
- **New** `updateBotUnreadIndicatorVisibility()` **method**: The new `updateBotUnreadIndicatorVisibility()` instance method specifies whether the unread indicator on the launcher icon is shown or hidden. For more information, see instance.updateBotUnreadIndicatorVisibility().
- Connect to Agent and custom cards now have rounded corners.
- Bug fixes.

6.6.2

Release date: 15 August 2022

- Bug fixes

6.6.1

Release date: 8 August 2022

- The `servers` property now supports a new `webChatScriptPrefix` option. Use this property to configure a proxy between your users' browsers and the IBM Cloud servers that host the web chat JavaScript code. For more information, see Setting up a proxy.

6.6.0

Release date: 25 July 2022

- A new `servers` property is now available in the web chat configuration options. You can use this property to set up a proxy between your users' browsers and watsonx Assistant. For more information, see Setting up a proxy.

6.5.2

Release date: 11 July 2022

- Bug fix for `date` response type.

6.5.1

Release date: 15 June 2022

- Bug fix for the Zendesk integration.

6.5.0

Release date: 6 June 2022

- **New agent events**: New events are now fired by the web chat when messages are sent or received during a conversation with a live agent using a service desk integration. For more information, see Agent events summary.
- Bug fixes.

6.4.1

Release date: 16 May 2022

- **Minimum size**: Reduced the minimum allowed size of the rendered web chat window to satisfy the accessibility requirements defined by the Web Content Accessibility Guidelines (WCAG) 2.1 standard.
- **Pop-up windows and tabs from iframes** : The web chat now allows pop-up windows and new tabs to be opened from content rendered inside `iframe` responses.
- **Faster responses**: Responses from the assistant are displayed more quickly and without a `...` typing indicator.

6.4.0

Release date: 18 April 2022

- **Date picker**: If you configure a step to collect a **Date** customer response, the step uses the new `date` response type to request that a graphical date picker displays so the customer can select a date, as an alternative to typing the date in the input field. Existing steps do not automatically inherit this behavior; if you want to use the date picker, you must delete the existing Date response and then re-add it.
- **Skip "connect to agent" card** : A new `serviceDesk.skipConnectAgentCard` configuration option is available. If this option is enabled, the web chat immediately connects to an agent when it receives a *Connect to agent* response, without first displaying a card and waiting for the user to click.
- **Close button**: A new `showCloseAndRestartButton` configuration option specifies whether the web chat interface shows an `X` (Close) button in addition to the existing `-` (Minimize) button. A customer can click this button to close the web chat, end the conversation, and end any conversation with a live agent. The chat transcript is also cleared, but any transcript of a conversation with a live agent is preserved.

6.3.0

Release date: 24 March 2022

- **Search cards**: Search cards have a new design.
- **New `restartConversation()` method**: The new `restartConversation()` instance method restarts the conversation with the assistant by clearing the web chat transcript and starting a new session. It also fires two new events (`pre:restartConversation` and `restartConversation`).

  For more information, see instance.restartConversation(), pre:restartConversation, and restartConversation.
- **New `agentEndConversation()` method**: The new `agentEndConversation()` instance method immediately ends the conversation with a live agent without requesting confirmation from the user. For more information, see instance.agentEndConversation().
- Bug fixes.

6.2.0

Release date: 7 March 2022

- **Navigation**: Added navigation features for web chat. For example, new “Back” and “Minimize” buttons make it easier to navigate between the home screen, the chat view, and panels. A new customizable drop-down menu appears near the avatar in both the assistant and agent chat views. Add new options to the menu of the web chat. For more information, see [updateCustomMenuOptions](https://web-chat.global.assistant.watson.cloud.ibm.com/docs.html?to=api-instance-methods#updatecustommenuoptions {: external}.

  The experience of connecting to a live agent is also improved to make it clearer how a user requests an agent, returns to chatting with the assistant, and ends the conversation.

  Animations for the web chat panels were improved to make the whole experience more seamless and cohesive.
- **Launcher**: The new web chat launcher bounces on two different occasions to attract attention and encourage customer engagement. For more information, see Launcher appearance and behavior.
- **Launcher**: Support added to control what text the launcher greets the user with, and when the greeting message is shown. For more information, see Launcher.
- **Locale**: The web chat no longer sets the locale in the system context to `en-us` when no locale is configured. The locale is set in the system context only if it is configured for web chat.
- Bug fixes.

6.1.0

Release date: 7 February 2022

- Updated to support internal changes to the preview link feature.

6.0.1

Release date: 24 January 2022

- Bug fix for the disclaimer. For more information, see Configuration options object.

6.0.0

Release date: 19 January 2022

- **API version**: The web chat now uses the `2021-11-27` version of the watsonx Assistant API. Previously it used the `2020-09-24` API version. For information about API changes introduced since the `2020-09-24` version, see the release notes for 27 November 2021 and 16 July 2021.
- **Launcher**: The new web chat launcher welcomes and engages customers so they know where to find help if they need it. For more information, see Launcher appearance and behavior.
- **Home screen**: Web chat home screen updated with a more modern look. For more information, see Configuring the home screen .
- **Agent events**: New events are fired when using a service desk integration and interacting with a live agent. If you use a custom service desk integration based on the starter kit, you can use these events to create a pre-chat form before the agent escalation occurs, to create a post-chat form after the agent conversation ends, or to specify what happens if an agent is not available (for example, create a ticket submission form). For more information, see Agent events summary.
- **Markdown support**: The web chat now fully supports common Markdown formatting in messages received from an assistant. You might need to review existing assistant output that contains strings that might be recognized as Markdown. (For example, a line of text that begins with a greater-

than (`>`) character is interpreted as a block quote.)

- **Time zone**: The time zone set in the context by the web chat no longer overrides any time zone set by the assistant.
- **Locale**: Any locale that is configured for the web chat is sent to the assistant as part of the context.
- **Window open events**: The `window:pre:open` and `window:open` events now fire any time that the chat window is opened, regardless of the reason. In previous releases, these events only fired if the window was opened by a customer clicking the built-in launcher. Other methods of opening the chat window, such as session history or custom launchers, did not fire these events.

  Event data passed to the listener has a new `reason` property that indicates the reason the window was opened. If you want to preserve the previous behavior, you can modify your handler to check this property:

  ```
  $ instance.on({ type: "window:open", handler: event => {
    if (event.data.reason === 'default_launcher') {
      // Previous code.
    }
  }});
  ```

  For more information, see Window open reasons.
- **hideCloseButton property renamed**: The `hideCloseButton` property for custom panels is renamed `hideBackButton`. The behavior of the property is unchanged. For more information, see customPanel.open().

5.1.2

Release date: 11 December 2021

- Bug fix for Salesforce integration.

5.1.1

Release date: 5 November 2021

- **"User is typing" support** : The web chat now supports displaying the "user is typing" message for service desks. This feature is supported for the Salesforce and Zendesk integrations, as well as any starter kit integration that implements it.
- Bug fixes.

5.1.0

Release date: 28 October 2021

- **Custom Panels**: The web chat now supports customizable panels that you can use to display any custom HTML content (for example, a feedback form or a multistep process). Your code can use instance methods to dynamically populate a custom panel, and open and close it. For more information, see Custom Panels.

5.0.2

Release date: 4 October 2021

- A new tutorial is now available that shows how to use Carbon components to customize user-defined responses and writeable elements.
- Bug fixes.

5.0.1

Release date: 20 September 2021

- Bug fixes.

5.0.0

Release date: 16 September 2021

- **New response types**: The web chat now supports the new `video` , `audio` , and `iframe` response types. For more information about these response types, see Rich responses.
- **Link to start web chat** : You can now create a set of HTML links that go directly to your web chat and start conversations on specific topics. For example, you might want to send an email to invite customers to update their account information; you can include a link that opens the web chat on

your site and sends the initial message `I want to update my account`. For more information, see Creating links to web chat.

- **CSS improvements**: Improved CSS to change the way the web chat resets styles in areas where you can include your own custom content, such as user-defined responses and writeable elements. The new approach better protects custom content from accidental style overrides. For more information, see Theming and custom content.

  > **Note:** If you have custom content (such as user-defined responses or writeable elements), verify that any styling is still rendering as you expect. Consider the new ibm-web-chat--default-styles class to maintain consistency with the web chat default styles.

- **Support for Carbon components**: As part of the new styling support, you can now use Carbon components in user-defined responses and web chat writeable elements. These components inherit any theming customizations that you made to the web chat.
- **New embedded script**: The embedded script that you use to add web chat to your website is updated to avoid unexpected code changes when you lock onto a web chat version. The previous version of the script continues to work but is now deprecated. If you want to upgrade your existing web chat deployments to use the new script, copy the updated code snippet from the **Embed** tab of the web chat integration settings. (Remember to reapply any customizations you made.)
- **Removal of deprecated methods and events** :
  - The `error` event is replaced by the `onError` method in the configuration object.
  - The `getID` method is removed.
- Microsoft Internet Explorer 11 is no longer a supported browser.

4.5.1

Release date: 30 August 2021

- Bug fixes for the interactive launcher beta feature. (For more information, see the `launcherBeta` at Configuration options object.)

4.5.0

Release date: 29 July 2021

- A new `scrollToMessage` method is available for scrolling the web chat view to a specified message in the chat history. For more information, see instance.scrollToMessage().
- A new `pre:open` event is available. This event is fired when the web chat window is opened, but before the welcome message or chat history are loaded. For more information, see window:pre:open.
- A new chat history widget is available for embedding in service desk agent UIs. This new widget is based on a read-only view of the standard web chat widget. For information about using the new chat history widget in integrations that are built with the starter kit, see Embedded agent application.

4.4.1

Release date: 6 July 2021

- Bug fixes.

4.4.0

Release date: 25 June 2021

- Bug fixes.

4.3.0

Release date: 7 June 2021

- **Search suggestions**: If a search skill is configured for your assistant, the suggestions include a new **View related content** section, which contains search results that are relevant to the user input.
- **Focus trap**: A new `enableFocusTrap` option enables maintaining focus inside the web chat widget while it is open. For more information, see Configuration options object.

4.2.1

Release date: 6 May 2021

- **Service URLs updated**: The URLs used by the web chat to communicate with the Assistant service were updated to remove the dependency on the deprecated `watsonplatform.net` domain. This change applies retroactively to version 3.3.0 and all subsequent web chat releases. Make sure the system that hosts the web chat widget has access to the new URL.

4.2.0

Release date: 27 April 2021

- **Conversation starters in suggestions** : The conversation starters that you configure for the home screen are now shown as suggestions. If suggestions are enabled, the conversation starters appear in a new section titled **People are also interested in** so customers can change the subject or start the conversation over.
- `onError` **callback**: The new `onError` callback option in the web chat configuration enables you to specify a callback function that is called if errors occur in the web chat; and makes it possible for you to handle any errors or outages that occur with the web chat. For more information, see Listening for errors.
- **Session ID available in widget state** : The state information returned by the `getState()` instance method now includes the session ID for the current conversation. For more information, see instance.getState().
- **IBM watermark**: The web chat can now display a **Built with IBM Watson** watermark to users. This watermark is always enabled for any new web chat integrations on Lite plans.
- **Fixes to rendering of list items** : Updated the rendering of HTML list items in the web chat widget.

4.1.0

Release date: 8 April 2021

- **Home screen now generally available** : Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to easily start chatting with the assistant.
- **Home screen enabled by default** : The home screen feature is now enabled by default for all new web chat deployments.
- **Home screen context support** : You can now access context variables from the home screen. Initial context must be set using a `conversation_start` node.

4.0.0

Release date: 16 March 2021

- **Session history now generally available**: Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. It is enabled by default. For more information about this feature, see Session history.

  Session history persists within only one browser tab, not across multiple tabs. The dialog provides an option for links to open in a new tab or the same tab. For more information, see this example on how to format links to open in the same tab.

  Session history saves changes that are made to messages with the pre:receive event so that messages still look the same on rerender. This data is only saved for the length of the session. If you prefer to discard the data, set `event.updateHistory = false;` so the message is rerendered without the changes that were made in the pre:receive event.

  instance.updateHistoryUserDefined() provides a way to save state for any message response. With the state saved, a response can be rerendered with the same state. This saved state is available in the `history.user_defined` section of the message response on reload. The data is saved during the user session. When the session expires, the data is discarded.

  Two new history events, history:begin and history:end announce the beginning and end of the history of a reloaded session. These events can be used to view the messages that are being reloaded. The history:begin event allows you to edit messages before they are displayed.

  For more information, see this example on saving the state of customResponse types in session history.

- **Channel switching**: You can now create a dialog response type to functionally generate a connect-to-agent response within channels other than web chat. If a user is in a channel such as Slack or Facebook, they can trigger a channel transfer response type. The user receives a link that forwards them to your organization's website where a connection to an agent response can be started within web chat.

3.5.0

Release date: 17 February 2021

- **Session history (beta)**: Web chat session history (beta) is now available. This feature makes it possible to maintain conversation history and context when customers refresh the page or navigate to a different page on the same website. For more information, see Session history (beta).

3.4.1

Release date: 2 February 2021

- Made an accessibility enhancement to the chat history. Now, you can use keys to navigate the messages by clicking the chat history, and pressing Enter and the arrow keys to move from one message to the next.
- Added the `instance.updateAssistantInputFieldVisibility()` to hide or show the text input field. For example, you might use the `pre:receive` event to check whether an options response type is returned and if so, hide the text field so the user is forced to pick one of the available options only.
- Added the `instance.getState()` method. You can use it to check for specific conditions, such as `isWebChatOpen`, before you perform an action that might rely on the condition being true.

For more information, see Instance methods.

3.3.2

Release date: 17 December 2020

- Addressed accessibility issues.

3.3.1

Release date: 3 December 2020

- The translated strings in the language files were revised and improved.
- An error message is shown now if a Java Web Token (JWT) that is provided with an incoming message is invalid. If the first JWT fails when the web chat opens, an error message is displayed in place of the web chat window that says, `There was an error communicating with watsonx Assistant`. If the initial JWT is valid, but the token for a subsequent message is invalid, a more discreet error message is displayed in response to the insecure input.
- Bug fixes.

3.3.0

Release date: 23 November 2020

- Added support for passing contextual information to a service desk agent from web chat.
- You can now customize a `user_defined` response type. For more information, see the Custom response type tutorial.
- Bug fixes.

3.2.1

Release date: 2 November 2020

- **Bug fix**: Fixing a bug that prevented the web chat integration preview from working after security was enabled.

3.2.0

Release date: 26 October 2020

- **Security improvement**: If you enable security, you no longer need to include the `identityToken` property when the web chat is loaded on a web page. If a token is not initially provided, the existing identityTokenExpired event fires when the web chat is first opened to obtain one from your handler.
- **Starter kit update**: Customize the timeout that occurs when the web chat integration checks whether any service desk agents are online.

3.1.1

Release date: 22 October 2020

- **Accessibility improvement**: Changed how the announcement text is generated to prevent announcements from being duplicated. Announcement text is hidden text that is provided for use by screen readers to indicate when dynamic web page changes occur.

3.1.0

Release date: 8 October 2020

- **Suggestions now allow for trial and error**: If customers select a suggestion and find that the response is not helpful, they can open the suggestions list again and try a different suggestion.

3.0.0

Release date: 22 September 2020

- **Choose when a link to support is included in suggestions** : The Suggestions beta feature has moved to its own tab. You can enable suggestions even if your web chat is not set up to connect to a service desk solution. You can control if and when the option to connect to customer support is available from the suggestions list.
- **Search result format change**: To support the ability to show more than 3 search results in a response, the search skill response type format changed. If you are using `pre:receive` or `receive` handlers to process search results, you might need to update your code. The `results` property was replaced by the `primary_results` and `additional_results` properties. For more information about the search skill response type format, see the API reference.
- **Language pack key change**: Due to improvements that were made to allow you to specify separate chat transfer messages for situations where agents are available and unavailable, the language source file was updated. The `agent_chatDescription` was renamed to `default_agent_availableMessage` and another key (`default_agent_unavailableMessage`) was added. If you defined a custom string for the `agent_chatDescription` key, you must modify your code to reflect this change.

2.4.0

Release date: 2 September 2020

- **Add a home screen**: Ease your customers into the conversation by adding a home screen to your web chat window. The home screen greets your customers and shows conversation starter messages that customers can click to submit to the assistant.

2.3.0

Release date: 10 August 2020

- **Introducing** *Suggestions*: Enable this beta feature to show a helper icon in the chat that customers can click to see alternate topics or to connect to an agent.
- **Search results are now expandable**: When search results are displayed in the web chat, users can click **Show more** to see more of the search result text.

2.2.0

Release date: 29 July 2020

- **Introduced the** `instance.writeableElements()` **method**: The `instance.writeableElements()` method gives you zones in the web chat user interface where you can embed your own content. For example, you can add content to the end of the header where it is displayed even as the chat content changes. Or you can add custom content that is displayed before the welcome node. For more information, see Instance methods.
- **Gave the** *Send* **button a new look**

  Changed the icon from  to  .
- **Securing your web chat is easier**: You no longer need to specify the `acr` claim when you define the JWT. The Authentication Context Class reference is managed by the web chat automatically.
- Improved the quality of non-English translations.
- Made a few minor bug fixes.

2.1.2

Release date: 2 July 2020

- **Loading issue**: Fixed an issue that prevented the web chat from loading properly for some deployments with short JWT expiration claims.

2.1.1

Release date: 1 July 2020

- **Service desk agent initials are displayed** : When the web chat transfers a user to a service desk agent, the agent's avatar is displayed in the chat

window to identify messages sent from the service desk agent. If the agent does not have an avatar, the first initial of the agent's name is displayed instead.

2.0

Release date: 16 June 2020

- **Versioning was added to web chat** : For more information, see Versioning.
- **Added bidirectional support**: You can now use the `direction` parameter to choose whether to show text and elements in the web chat in right-to-left or left-to-right order. For more information, see Configuration.
- **Introduced the `instance.destroySession()` method**: The `instance.destroySession()` method removes all cookie and browser storage that is related to the current web chat session. You can use this method as part of your website logout sequence. For more information, see Instance methods.

1.5.3

Release date: 14 April 2020

- **The *Font family* field was removed from the configuration page** : The text that is displayed in the chat window uses the fonts: `IBMPlexSans, Arial, Helvetica, sans-serif` . If you want to use a different set of fonts, you can customize the CSS for your web chat implementation. For more information, see Theming.
- When your implementation does not specify a unique user ID, the web chat adds a first party cookie with a generated anonymous ID to use to identify the unique user. The generated cookie now expires after 45 days. For more information, see GDPR and cookie policies.

1.5.2

Release date: 2 April 2020

- **Introduced the `learningOptOut` parameter**: You can add the `learningOptOut` parameter and set it to `true` to add the `X-Watson-Learning-Opt-Out` header to each `/message` request that originates from the Web Chat. For more information about the header, see Data collection. For more information, see Configuration.

1.4.0

Release date: 20 March 2020

- **Customize the CSS theme**: For more information, see Theme configuration.
- **Shadow DOM is no longer used** : When you use custom response types or HTML in your dialog, you can apply CSS styles that are defined in your web page to the assistant's response. To override any default styling in the web chat, you must specify the `!important` modifier in your CSS.

Release notes for watsonx Assistant for IBM Cloud Pak for Data

IBM Cloud Pak for Data

Use these release notes to learn about the latest updates to watsonx Assistant for IBM Cloud Pak® for Data.

For more information on the latest updates of watsonx Assistant On-premises version 5.1.0 and above, see IBM® Software Hub.

Web chat versions

When you install watsonx Assistant for IBM Cloud Pak® for Data, the latest available version of the web chat integration is included. See the following table for details about the latest available web chat version for each watsonx Assistant for IBM Cloud Pak® for Data release. If your web chat version isn't locked, then the web chat integration is upgraded to the latest available version when you upgrade watsonx Assistant for IBM Cloud Pak® for Data.

The following table shows the latest version of the web chat integration that is included with each release of watsonx Assistant for IBM Cloud Pak® for Data. IBM Cloud Pak® for Data supports web chat versions 5.1.1 or later. To customize and change version numbers, see Controlling the web chat version.

watsonx Assistant for IBM Cloud Pak® for Data version	Latest web chat version available
5.2.1	8.11.0
5.2.1	8.10.0

5.0.3	8.2.6
5.0.1	8.1.2
5.0.0	8.1.0
4.8.9	8.5.1
4.8.8	8.5.1
4.8.7	8.3.1
4.8.6	8.2.3
4.8.5	8.0.0
4.8.4	7.10.0
4.8.3	7.9.0
4.8.2	7.8.0
4.8.0	7.7.0
4.7.4	7.6.0
4.7.2	7.5.0
4.7.1	7.4.0
4.7.0	7.3.0
4.6.5	7.2.0
4.6.3	7.1.0
4.6.2	7.0.0
4.6.0	6.7.0
4.5.3	6.6.2
4.5.1	6.5.2
4.5.0	6.4.1
4.0.8	6.2.0

Web chat versions in watsonx Assistant for IBM Cloud Pak® for Data

Version 4.8.9 (30 April 2025)

Updates

No new features or updates.

Version 4.8.8 (29 January 2025)

Minor updates

Added known issues and updated upgrade sections.

Version 4.8.7 (30 October 2024)

Minor updates

Added known issues.

Version 5.0.3 (27 September 2024)

Configure security certificates

You can now secure the search integrations, custom extensions, conversational skills, and webhooks by configuring TLS certificates in your assistant. For more information, see Configuring Security certificates.

Use custom service integration to search information

You can now use a custom service integration to create your own search capability in your assistant that uses conversational search for enhanced AI responses. For more information, see Custom service integration.

Tune the generated response length in conversational search

When you use conversational search in your assistant, you can now define the length of the generated responses that your assistant gives to better fit your needs. For more information, see Conversational search.

Version 4.8.6 (28 August 2024)

Minor updates

Added known issues.

Version 5.0.1 (31 July 2024)

Updates

The autocorrection feature is now disabled for search integration in your assistant. For more information, see Auto correction

Version 5.0.0 (19 June 2024)

Conversational search

The new conversational search feature has a built-in retrieval-augmented generation(RAG) solution that helps your assistant to extract an answer from the highest-ranked query results and returns a text response to the user. For more information, see Conversational search.

Integration of Elasticsearch to the search feature

You can now integrate Elasticsearch to the search feature in your assistant. With Elasticsearch, your assistant can perform different types of searches such as metric, structured, unstructured, and semantic with higher accuracy and relevance by making use of enterprise content. The data analytics engine in Elasticsearch expands the scope of search integration to larger data sets in assistant. For more information about Elasticsearch search integration, see Elasticsearch search integration setup.

Tuning your assistant's tendency to say "I don't know"

You can now tune the tendency of your assistant to say "I don't know" in conversational search by using the **Tendency to say “I don’t know”** option in the conversational search settings. This option can help to reduce Large Language Model (LLM) hallucinations and provide higher fidelity answers for conversational search by tuning your assistant's tendency to fall back to the “I don’t know” answer. For more information, see Tuning conversational search’s tendency to say “I don’t know".

Streaming response for conversational search

You can now use streaming response in your assistant for conversational search. With the help of watsonx.ai capabilities, streaming response can provide continuous and real-time responses. For more information, see Streaming response.

Overwrite all or skip all when you copy actions to another assistant

You can now choose to overwrite all references or skip all references when you copy actions from one assistant into another. For more information, see Copying an action to another assistant.

Add a custom result filter for the IBM Watson® Discovery search integration

You can now filter your search result in the IBM Watson® Discovery search integration by adding custom text strings in the Custom result filter field in Search integration. For more information, see Configure the search for IBM Watson® Discovery.

Configure search routing

You can configure the search routing for your assistant when no matches are available for the customer response. For more information, see Configuring the search routing when no action matches.

Conversational skills

You can now use conversational skills in your assistant to begin tasks or workflows. You must register a pro code conversational skill provider on your assistant instance and begin building skill-backed actions to fit your use cases. For more information, see Conversational skills API documentation.

Service monitors

Your assistant can now use service monitors to monitor the health of your assistant instances. For more information, see Installing service monitors.

29 March 2024

Algorithm version Latest(15 Apr-2023) uses improved intent detection and matching

The **Latest(15 Apr-2023)** algorithm version uses a new foundation model to improve the intent detection and action matching in assistants with languages such as English, French, German, Portuguese (Brazilian), Spanish, Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, Italian, Japanese, and Korean. The new foundation model is trained by using the transformer architecture. For more information, see Algorithm version and training.

Change in backup PersistentVolumeClaim (PVC) name

The backup PersistentVolumeClaim (PVC) name is changed from store-pvc to store-db-backup-pvc. For more information, see Backing up and restoring data.

Change in the default size of Postgres backup PersistentVolume (PV)

The default size of Postgres backup PersistentVolume (PV) is now 10Gi. For more information, see Backing up and restoring data.

31 January 2024

Mask confidential customer information

You can now mask confidential customer information in an action by marking variables as private. For more information, see the following topics: Using variables to manage conversation information Protecting the privacy of the customer information

Activity log

You can use the activity log to track changes and modification that you made to an assistant. For more information, see Activity log.

Message logs for a dialog without action

You can now use the Message logs tab on the Analyze page to see a log of all messages that are sent between the user and the assistant by turns. This feature is available for all messages, including the messages that are sent when dialog is activated. For more information, see Review customer conversations.

More control over pause responses

Use a pause response to have your assistant wait for a specified interval before the next response. Previously, pauses were 1 to 10 seconds in length. You can now pause a response for 0 to 60 seconds. You can also specify milliseconds by specifying a decimal. For more information, see Pause response.

Visualize an action

After you create an action, you can optionally switch from the step edit view to a visualization that displays a canvas with a flowchart of the action. For more information, see Visualizing the flow of the action.

Invalid date entry not accepted

Starting from this release, the assistant does not recognize invalid date entries such as Feb 31, 31/11/2022, and Feb 29 2023. In addition, the assistant does not automatically parse the invalid dates to the first day of the month. For more information, see @sys-date.

OpenAPI document file size limit for integrating custom extension

When you integrate a custom extension by using REST API, the maximum file size of the OpenAPI document that you can import is limited to 4 MB. For more information, see Preparing the API definition.

29 November 2023

IBM Watson® Assistant is now IBM® watsonx™ Assistant.

IBM Watson® Assistant was renamed to IBM® watsonx™ Assistant.

Store your custom metrics in watsonx Assistant

You can store your custom metrics for assistants by using the metrics method. For more information, see Monitoring the platform.

Diagnostic logs

You can now use the diagnostics logs in IBM Cloud Pak® for Data for your assistants. For more information, see diag.

25 October 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.7.4 is available

Watson Assistant for IBM Cloud Pak® for Data 4.7.4 is compatible with IBM Cloud Pak® for Data Version 4.7. See the support matrix for more details.

Session history summary

You can use the session_history variable to store a summary of the recent messages from a conversation for each customer. You might use session history to provide a summary of the conversation during a transfer to a live agent, or you might use it to call a generative AI extension to generate an answer. For more information, see Session history.

Specify how often to use the No matches action

You can use a new global setting for actions to change how often your assistant routes customers to the `No matches` action. By setting this threshold, you can specify when the assistant fetches answers from a search integration, triggers the "Fallback" action, or switches topics. For more information, see When the assistant can't understand your customer's request.

See who last edited a collection or action

Now you can see who last edited a collection or action. On the Actions page, you can hover on the values in the Last edited column to see the email address of the person who last modified the collection or action.

Change to multilingual downloads for translation

The ID values that are used in the multilingual downloads for translation are changed. If you used the multilingual download before, you need to download a new CSV file to match the IDs in your assistant. For more information, see Using multilingual downloads for translation.

Algorithm version **Beta** provides improved intent detection and action matching for more languages

The algorithm version **Beta** now provides improved intent detection and action matching for Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, Italian, Japanese, and Korean. It includes a new foundation model that is trained using a transformer architecture to improve intent detection and action matching.

Improvements include:

- Improved robustness to variations in user inputs such as typos and different inflection forms
- Less training data required to reach the same level of performance compared to previous algorithms

For more information, see Algorithm version and training

30 August 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.7.2 is available

Watson Assistant for IBM Cloud Pak® for Data 4.7.2 is compatible with IBM Cloud Pak® for Data Version 4.7. See the support matrix for more details.

Multiple validation responses

When you edit a validation for a customer response, you can now include several validation responses. For more information, see Customizing validation for a response.

Multilingual downloads

You can download language data files, in CSV format, so that you can translate training examples and assistant responses into other languages for use in other assistants. For more information, see Using multilingual downloads for translation.

Algorithm for improved intent detection and action matching

You can now use the algorithm version Beta, which includes a new foundation model that is trained with a transformer architecture. The Beta algorithm provides the following improvements:

- Improved intent detection and action matching for English, French, German, Portuguese (Brazilian), and Spanish
- Improved robustness to variations in user inputs such as typos and different inflection forms
- Reduction in the amount of training data required to reach the same level of performance compared to previous algorithms

For more information, see Algorithm version and training.

Fallback choice for dynamic options

The dynamic options response type now includes a fallback static choice, such as "None of the above", if the options aren't what the customer wants. You can then add a step that is conditioned on this static option to provide further assistance. For more information, see Dynamic options.

Option to allow change of topic between actions and dialog

If you are using actions and dialog, there is a new setting you can use to ensure that customers can change topics between an action and a dialog node. For more information, see Allow change of topic between actions and dialog.

Action and collection names must now be unique

With this release, each action name must be different from other action names, and each collection name must be different from other collection names. If your existing actions or collections have duplicate names, a warning icon will appear in the Status column. For more information, see Overview: Editing actions and Organizing actions in collections.

26 July 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.7.1 is available

Watson Assistant for IBM Cloud Pak® for Data 4.7.1 is compatible with IBM Cloud Pak® for Data Version 4.7. See the support matrix for more details.

Edit step titles

You can now add and edit titles for each step, which can help you more easily identify what a step does in an action. For more information, see Editing actions.

Filtering the list of actions

You can locate specific actions by filtering the list by subactions, by custom extension, or by variable. For more information, see Filtering actions.

See which actions use a specific variable

The Variables page now includes a new *Actions count* column. You can click on the number in the column to see which actions use a variable. For more information, see Creating a session variable.

New expression choice for setting a session variable

Previously, to use an expression to set or modify a variable value, you needed to pick an existing variable or create a new one and select the expression option. Now you can use a new **Expression** choice to write an expression directly without first picking a variable. For more information, see Storing a value in a session variable.

Changes to validation of OpenAPI specifications

When you build a custom extension, you work with OpenAPI specification files. This release includes changes to the validation of OpenAPI files, which might affect the connection between your actions and extensions

You can import an action with references to a custom extension and Watson Assistant can automatically connect the action and extension. With the validation change, modifications to the OpenAPI specification for your custom extension might affect this automatic connection. For more information, see Building a custom extension.

Changes to the date and number formats in assistant responses

You might see changes to the date and number formats in assistant responses.

Examples of date changes include removed or added periods, such as:

- In Spanish, `18 abr. 2021` changes to `18 abr 2021`
- In Portuguese, `18 de abr` changes to `18 de abr.`

The delimiter character changes for numbers in some languages. For example, in French, nonbreaking space (NBSP) changes to narrow no-break space (NNBSP).

These changes are the result of migrating the Watson Assistant platform to Java 17, where locale values are updated by using specifications in CLDR 39.

To avoid or minimize the impact of similar changes in the future, you can use Display formats.

Differences in contextual entity detection for dialog skills with few annotations

If you have 10 to 20 examples of contextual entities in your dialog skill, you might see differences in the entities detected due to updates made to address critical vulnerabilities. The impact of these differences is limited to only newly-trained models. Existing models are unaffected. You can mitigate these differences by annotating more examples. For more information, see Annotation-based method.

28 June 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.7.0 is available

Watson Assistant for IBM Cloud Pak® for Data 4.7.0 is compatible with IBM Cloud Pak® for Data Version 4.7. See the support matrix for more details.

The new experience is now available for all new instances

When you create a new instance, the new experience will be the default interface to use for building your assistants. The new experience makes it easier to use actions to build customer conversations. If you don't want to use the new experience, you can use the Manage menu to switch to the classic experience. For details, see Welcome to the new watsonx Assistant.

All languages are now enabled by default

You don't need to add languages during installation. All supported languages are now enabled by default with no increase in footprint. For details, see Supported languages.

New algorithm version Latest (20 Dec 2022) provides improved irrelevance detection

A new algorithm version is available. The Latest (20 Dec 2022) version includes a new irrelevance detection implementation to improve off-topic detection. For details, see Algorithm version and training.

Actions templates updated with new design and new choices

The actions template catalog has a new design. Now you can select multiple templates at the same time. The catalog also has new and updated

templates, including starter kits that you can use with external services such as Google and HubSpot. For details, see Building actions from a template.

Organize actions into collections

Now you can put actions into collections, which are folder-style groups based on whatever categorization you need at your organization, such as by use case, internal team, or status. For details, see Organizing actions in collections.

Display iframe inline in the conversation

In the web chat, an assistant can now include an iframe response within the conversation. This new option is useful for smaller pieces of iframe content. For details, see Adding an iframe response.

New validation choices for date, time, and numeric customer responses

For Number, Date, Time, Currency, and Percentage customer responses, you can now customize the validation to check for a specific answer, such as a range of dates or a limited currency amount. For details, see Customizing validation for a response.

Confirmation to return to previous action

If a customer changes to a different topic, assistants now ask a "yes or no" confirmation question to determine whether the customer wants to return to the previous action. Previously, the assistant returned to the previous action without asking. New assistants are set to use this confirmation by default. For details, see Confirmation to return to previous topic.

New "Never return" choice for when a customer changes topics

In some cases, you might not want a customer to return to a previous action after the customer changes the topic. To set up this option, use the new Never return choice in Action settings. For details, see Disabling returning to the original topic.

Allow changing topics in free text and regex responses

By default, customers can't change topics when the assistant is asking for a free text response or when an utterance matches the pattern in a regex response. Now free text and regex customer response types have a setting to allow a customer to digress and change topics. For details, see Enabling changing the topic for free text and regex customer responses.

Adding and using multiple environments

Each assistant has a draft and live environment. You can now add up to three environments to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments. For details, see Adding and using multiple environments.

Display formats for variables

In the Global settings page for actions, you can use the Display formats tab to specify the display formats for variables that use date, time, numbers, currency, or percentages. You can also choose a default locale to ensure that the format of a variable that's displayed in the web chat is what you want for your assistant. For example, you can choose to have the output of a time variable appear in HH:MM format instead of HH:MM:SS. For details, see Display formats.

Debug custom extensions

You can use the new extension inspector in the action editor Preview pane to debug problems with custom extensions. The extension inspector shows detailed information about what data is being sent to and returned from an external API. For details, see Debugging failures.

New expression choice for setting a session variable

Previously, to use an expression to set or modify a variable value, you needed to pick an existing variable or create a new one and select the expression option. Now you can use a new Expression choice to write an expression directly without first picking a variable. For details, see Storing a value in a session variable.

Using the Cloud Object Storage importer to migrate chat logs

You can use the Cloud Object Storage importer service to migrate your chat logs from one installation of Watson Assistant to another.For more information, see Using the Cloud Object Storage importer to migrate chat logs.

2 May 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.6.5 is available

Watson Assistant for IBM Cloud Pak® for Data 4.6.5 is compatible with IBM Cloud Pak® for Data Version 4.6. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

Algorithm version 01-Jun-2022 uses enhanced intent detection by default

The algorithm version **Latest (01-Jun-2022)** now uses enhanced intent detection by default. Before this change, some skills that did not include a specific algorithm version selection inadvertently used **Previous (01-Jan-2022)**. You can notice small changes in intent detection behavior when changes are made to an assistant that previously didn't have enhanced intent detection enabled. For more information, see Algorithm version and training.

Test the **Beta** algorithm version now to prepare for release 4.7.0

The **Beta** algorithm version in release 4.6.5 includes a new irrelevance detection implementation to improve off-topic detection accuracy.

Improvements include:

- Relevant user inputs are expected to get higher confidence, so they are less likely to be considered irrelevant or require clarification
- Irrelevance detection is improved in the presence of direct entity references
- Irrelevance detection is more stable across small changes to input
- Intent detection is more stable regarding occurrence of numerics, such as postal codes
- For German-language assistants, intent detection is more robust in the presence of umlauts

In the forthcoming 4.7.0 release, it is planned that this **Beta** version will become the **Latest** version, replacing the current **Latest (01 Jun 2022)** version. You can test the **Beta** version in 4.6.5 now to prepare.

For more information, see Algorithm version and training.

23 February 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.6.3 is available

Watson Assistant for IBM Cloud Pak® for Data 4.6.3 is compatible with IBM Cloud Pak® for Data Version 4.6. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

30 January 2023

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.6.2 is available

Watson Assistant for IBM Cloud Pak® for Data 4.6.2 is compatible with IBM Cloud Pak® for Data Version 4.6. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

New Watson Assistant v2 APIs

Watson Assistant now provides new methods related to assistants, skills, environments, and releases. For details, see the Watson Assistant v2 API documentation.

30 November 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.6.0 is available

Watson Assistant for IBM Cloud Pak® for Data 4.6.0 is compatible with IBM Cloud Pak® for Data Version 4.6. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

New watsonx Assistant experience

With the new experience in watsonx Assistant, you can build, test, publish, and analyze your assistant from one simple and intuitive interface that focuses on using actions to build customer conversations. To learn more about building your assistant with actions, see Algorithm version.

Credential rotation

You can now rotate your credentials for added data store security. See Managing security for your Watson Assistant datastores.

Autoscaling

Using Red Hat® OpenShift® Horizontal Pod Autoscaler, you can optionally enable autoscaling for Watson Assistant so that the service dynamically scales up and down to make efficient use of cluster resources. For details, see Automatically scaling resources for services.

New commands for shutting down and restarting services

The `cpd-cli` manage command now includes `shutdown` and `restart` sub-commands to make it easier to shut down and restart the IBM Cloud Pak® for Data services on your cluster. For details, see Shutting down and restarting services.

Action skills

The classic Watson Assistant experience now supports action skills.

13 October 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.5.3 is available

Watson Assistant for IBM Cloud Pak® for Data 4.5.3 is compatible with IBM Cloud Pak® for Data Version 4.5. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

Support for `autoScaleConfig`

Support for automatically scaling resources using the Red Hat OpenShift Horizontal Pod Autoscaler (HPA). See Automatically scaling resources for services.

3 August 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.5.1 is available

Watson Assistant for IBM Cloud Pak® for Data 4.5.1 is compatible with IBM Cloud Pak® for Data Version 4.5. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

New deployment sizes

Two new deployment sizes are now available. The two new sizes are `Starter` and `Production`. The `Starter` size is equivalent to the `small` deployment size, and the `Production` size is equivalent to the `medium` deployment size.

29 June 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data Version 4.5.0 is available

Watson Assistant for IBM Cloud Pak® for Data 4.5.0 is compatible with IBM Cloud Pak® for Data Version 4.5. See the support matrix for more details. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

Red Hat OpenShift Container Platform support

You can deploy IBM Cloud Pak® for Data Version 4.5 on the following versions of Red Hat OpenShift Container Platform:

- Version 4.6.29 or later fixes
- Version 4.8.0 or later fixes
- Version 4.10.0 or later fixes

New IBM Cloud Pak® for Data CLI commands and reference

Starting in IBM Cloud Pak® for Data Version 4.5, the `cpd-cli` includes new commands and a new command reference. For more information, see the cpd-cli command reference.

Microsoft Edge browser support

Beginning in Version 4.5, Microsoft Edge is fully supported for use with IBM Cloud Pak® for Data.

Language support improvements

Entity recognition and intent classification for Japanese and Korean languages changed to improve the reliability of Watson Assistant. You might see minor differences in how Watson Assistant handles entity recognition and intent classification.

Any visible changes are most likely to be seen in dictionary-based or pattern-based entity matching. For more information about defining entities, see

Adding entities. As a suggested practice, you can test your dialog skill with your current test framework to determine whether your workspace is impacted before you update your production workspace.

If entity values or synonyms that previously matched no longer match, you can update the entity and add a synonym with white space between the tokens, for example:

- Japanese: Add “見 た” as a synonym for “見た”
- Korean: Add “잘 자 요” as a synonym for “잘자요”

Assistant preview link can be disabled

Assistant preview now includes a toggle to disable the preview link. This allows you to stop access to the preview link if necessary.

27 April 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.8 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.8 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

Closed entity matching with accent-normalized values in French

Closed entities exact matches in French are completed using accent-normalized values or synonyms. For example, if you define a closed entity with a value or synonym with accent marks (for example, garçon or déjà), then variants without accent marks are also recognized (garcon or deja). Likewise, if a closed entity value or synonym is defined without accent marks, then user inputs with accent marks are also recognized. For more information about defining entities, see Defining information to look for in customer input.

Pattern entities do not prevent spelling autocorrection

Pattern entities that match all characters and words that are usually used to count input words do not prevent spelling autocorrection. For example, if a customer defines the `^..{0,19}$` pattern entity that matches the first 20 characters of an input, then the entity match does not affect spelling autocorrection. In this example, an input of `cancl transaction` is autocorrected to `cancel transaction`.

This change applies to the following languages: English and French. For more information, see Autocorrecting user input.

Fuzzy matching updates

Previously, an update was made so that interactions between the stemming and misspelling fuzzy matching features were not allowed. This change applied to the following languages: English, French, German, and Czech. This was updated so that this change applies only to the English language. For more information, see How fuzzy matching works.

Improved irrelevance detection for Dutch

Irrelevance detection for Dutch disregards any punctuation in an input sentence. For example, you can now expect the same confidence score for the following two inputs: `ik ben een kleine krijger?` and `ik ben een kleine krijger`. In this example, the question mark (`?`) doesn't affect the confidence score.

Improved enhanced intent detection

The exact match in enhanced intent detection now better handles small differences between training examples and runtime utterances when the differences do not change the meaning of a sentence.

For example, suppose in your training examples, `covid-19` is in the `#covid` intent and `@doctortype_facilitytype around Palm Beach` is in the `#find_provide_master` intent. In this example, the `@doctortype_facilitytype` direct entity reference contains entity values, including `hospital`. At run time, `covid19` is predicted as 100% confident for the `#covid` intent, and `hospital around palm beach` is predicted as 100% confident for the `#find_provide_master` intent.

This update applies to the following languages: English, French, Spanish, Italian, and the universal language model. For more information, see Accessing intents.

30 March 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.7 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.7 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

Enhanced irrelevance detection update

We revised the enhanced irrelevance detection classification algorithm. Now, enhanced irrelevant detection uses any provided counterexamples in training. This change does not affect workspaces without counterexamples. This update applies to the following languages: English, French, Spanish, Italian, and the universal language model. For more information, see Defining what's irrelevant.

Fuzzy matching updates

Interactions between the stemming and misspelling fuzzy matching features are not allowed. Improve fuzzy matching behavior by limiting the interactions between different fuzzy matching features. This change applies to the following languages: English, French, German, and Czech. For more information, see How fuzzy matching works.

23 February 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.6 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.6 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes and features.

New API version

The current API version is now `2021-11-27`. This version introduces the following changes:

- The `output.text` object is no longer returned in `message` responses. All responses, including text responses, are returned only in the `output.generic` array.

Configure webhook timeout

From the **Pre-message webhook** and **Post-message webhook** configuration pages, you can configure the webhook timeout length from a minimum of 1 second to a maximum of 30 seconds. For more information, see Webhook overview.

Rich response types

Your assistant can now send responses that include elements such as audio, video, or embedded `iframe` content. For more information, see Rich responses.

Analytics overview change

To improve reliability, the **Values** column has been removed from **Top entities** on the **Analytics Overview** page. **Top Entities** continues to provide counts of entity types. For more information, see Top intents and top entities.

Dialog skill **Try it out** improvements

For dialog skills, the **Try it out** pane now uses the React UI framework similar to the rest of the Watson Assistant user interface. You shouldn't see any change in behavior or functionality. As a part of the update, dialog skill error handling has been improved within the **Try it out** pane.

Disambiguation feature updates

The dialog skill disambiguation feature now includes improved features:

- **Increased control**: The frequency and depth of disambiguation can now be controlled by using the **sensitivity** parameter in the workspace API. There are 5 levels of sensitivity:
  - `high`
  - `medium_high`
  - `medium`
  - `medium_low`
  - `low`

  The default (`auto`) is `medium_high` if this option is not set.
- **More predictable**: The new disambiguation feature is more stable and predictable. The choices shown may sometimes vary slightly to enable learning and analytics, but the order and depth of disambiguation is largely stable.

These new features may affect various metrics, such as disambiguation rate and click rates, as well as influence conversation-level key performance

indicators such as containment.

If the new disambiguation algorithm works differently than expected for your assistant, you can adjust it using the sensitivity parameter in the update workspace API. For more information, see Update workspace.

26 January 2022

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.5 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.5 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8. This release of Watson Assistant for IBM Cloud Pak® for Data includes various fixes. Watson Assistant for IBM Cloud Pak® for Data 4.0.5 is compatible with FIPS-enabled clusters.

20 December 2021

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.4 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.4 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8. This release of Watson Assistant for IBM Cloud Pak® for Data includes critical security fixes. Watson Assistant for IBM Cloud Pak® for Data 4.0.4 is not compatible with FIPS-enabled clusters. The upcoming release of Watson Assistant for IBM Cloud Pak® for Data 4.0.5 will be compatible with FIPS-enabled clusters.

4 October 2021

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.2 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.2 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.8.

Integration with the IBM Cloud Pak® for Data auditing service

Watson Assistant integrates with the IBM Cloud Pak® for Data auditing service feature, providing standard auditing records for important lifecycle and security events. The service generates audit records for events such as intent edits, entity creation, dialog node deletion, and more.

29 July 2021

IBM Watson® Assistant Cartridge for IBM Cloud Pak® for Data 4.0.0 is available

Watson Assistant for IBM Cloud Pak® for Data 4.0.0 is compatible with IBM Cloud Pak® for Data 4.0 on Red Hat OpenShift 4.6. See the support matrix for more details.

Universal language

You now can build an assistant in any language you want to support. If a dedicated language model is not available for your target language, create a skill that uses the universal language model. The universal model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from training data written in the target language that you add to it.

Premessage, postmessage, and log webhooks

A set of new webhooks are available for each assistant. You can use the webhooks to perform preprocessing tasks on incoming messages and postprocessing tasks on the corresponding responses. You can use the new log webhook feature to log each message with an external service.

Features not included

This release does not include the following features, which are available for cloud instances at the time of this release:

- Service desk support is not included.
- The phone integration is not supported.
- The actions skill is not available.
- The intent recommendations feature and the enhanced intent detection model are not supported.

19 March 2021

IBM Watson® Assistant 1.5.0 patch 1 is available

For installations on IBM Cloud Pak® for Data 3.5, patch 1 includes configuration changes for FIPS compatibility and other fixes. See Available patches for IBM Watson® Assistant for IBM Cloud Pak® for Data

9 December 2020

IBM Watson® Assistant for IBM Cloud Pak® for Data 1.5.0 is available

Watson Assistant for IBM Cloud Pak® for Data 1.5.0 is compatible with IBM Cloud Pak® for Data 3.5 and IBM Cloud Pak® for Data 3.0.1 deployments on Red Hat OpenShift 4.6, 4.5, or 3.11. See the support matrix for more details.

IBM Watson® Assistant for IBM Cloud Pak® for Data 1.5.0 is available

Watson Assistant for IBM Cloud Pak® for Data 1.5.0 is compatible with IBM Cloud Pak® for Data 3.5 and IBM Cloud Pak® for Data 3.0.1 deployments on Red Hat OpenShift 4.5 or 3.11.

Access conversation logs and metrics

An **Analytics** page is now available. From this page, you can see conversation logs and metrics that give you insights, such as what topics your customers are asking about and how often the assistant succeeds in addressing customer requests. For more information, see Metrics overview.

Introducing the *Web chat* integration

Deploy your assistant in minutes. Create a web chat integration to embed your assistant into a page on your website as a chat widget. Web chat version 3.3.0 is included in this release.

Introducing the *Preview link* integration

Deploy your assistant to an IBM-branded web site for testing purposes. When you add both a dialog skill and search skill to your assistant, you can test the overall interaction between the two skills by using the preview link integration.

Improved system entities

A new `interpretation` property is returned for the system entities in all languages. The new property provides additional information about the recognized entity which can be leveraged by your dialog. For more information, see System entities.

Irrelevance detection

The irrelevance detection classification algorithm was updated to make it even smarter out of the box. Now, even before you begin to teach the system about irrelevant requests, it is able to recognize user input that your skill is not designed to address. The new algorithm is enabled by default for any English-language skills that you import or create. For more information, see Irrelevance detection.

Search was added to the Dialog, Intents, and Entities pages

You can now search within the product. For example, if you want to find any dialog nodes that condition on an intent, you can open the Dialog page and search on the intent name.

v2 Logs API is available

Use the v2 API logs method to list log events for an assistant. For more information, see the API reference documentation.

Features not included

This release does not include the following features, which are available for cloud instances at the time of this release:

- While the web chat integration is now available, service desk support is not included.
- The Facebook, Slack, Intercom, Phone, SMS with Twilio, and WhatsApp integrations are not supported.
- The actions skill is not available.
- You cannot enable FAQ extraction when you add a web crawl data collection to the search skill.
- You cannot use the Activity Tracker service to track user actions for auditing purposes.
- You cannot manage user access at the individual skill and assistant level. You can control only who can access the entire service instance, which includes all of its skills and assistants. For more information about granting access to services in IBM Cloud Pak for Data, see 3.5 Managing users.

19 June 2020

IBM Watson® Assistant for IBM Cloud Pak® for Data 1.4.2 is available

Watson Assistant for IBM Cloud Pak® for Data 1.4.2 is compatible with IBM Cloud Pak® for Data 3.0.1 on OpenShift Red Hat 3.11 or 4.5 and IBM Cloud Pak® for Data 2.5 deployments on OpenShift Red Hat 3.11. The 1.4.2 release is certified on Red Hat OpenShift 4.5.

Autocorrection support was added

Autocorrection helps your assistant understand what your customers want. It corrects misspellings in the input that customers submit before the input is evaluated. With more precise input, your assistant can more easily recognize entity mentions and understand the customer's intent. This feature is not available in all languages. See Autocorrecting user input for details.

French-language dialog skill improvements

Added full support for fuzzy matching and contextual entities and added beta support for autocorrection.

The Covid-19 content catalog is available in Brazilian Portuguese, English, French, and Spanish

The content catalog defines a group of intents that recognize the common types of questions people ask about the novel coronavirus. You can use the catalog to jump-start development of chatbots that can answer questions about the virus and help to minimize the anxiety and misinformation associated with it. For more information about how to add a content catalog to your skill, see Using content catalogs.

Backup automation

Support was added for doing data backups on a schedule.

Stateless v2 message API

The v2 runtime API now supports a new stateless `message` method. For more information, see the API Reference}.

Architecture improvements

The Skill-conversation microservice was removed. The microservice used to convert v2 API calls to v1 format and the other way around. The conversion is now done within the Store microservice. Reimplementing this function in the Store increased the overall speed with which v2 API requests are processed. The Spellchecker and CLU Embedding microservices were added. The word embeddings that are used by the language understanding pipeline (training, TAS, ED-MM) now are stored in the CLU Embedding microservice instead of MongoDB.

Slot prompt JSON editor

You can now use the context or JSON editors for the slot response field where you define the question that your assistant asks to get information it needs from the customer. For more information about slots, see Gathering information with slots.

Bug fixes

Multiple bug fixes were made.

Features not included

This release does not include the following features, which are available for cloud instances at the time of this release:

- There are no metrics or analytics capabilities. Therefore, the *Analytics* page is not included in the product user interface.
- There are no deployment connectors or built-in integrations available. You must build a custom client application that can host the assistant. As a result, the *Integrations* page is not included in the product user interface.
- The @sys-person and @sys-location system entities are not supported and the new version of the numeric system entities is not available.
- There is no search function in the pages of the product.
- You cannot use the Activity Tracker service to track user actions.
- You cannot manage user access at the individual skill and assistant level. You can control only who can access the entire service instance, which includes all of its skills and assistants.

14 April 2020

IBM Cloud Private End Of Support

Effective 30 September 2020, IBM will withdraw support for IBM Watson® Assistant on IBM Cloud Private. For more information, see the Product lifecycle page.

28 February 2020

IBM Watson® Assistant for IBM Cloud Pak® for Data version 1.4.1 is available

Watson Assistant for IBM Cloud Pak® for Data version 1.4.1 is compatible with IBM Cloud Pak® for Data version 2.5.

New backup and restore

A new backup and restore script is available.

Webhooks now generally available

The Webhooks feature is now generally available. See Making a programmatic call from dialog.

Bug fixes

Multiple bug fixes were made.

Features not included

This release does not include the following features, which are available for cloud instances at the time of this release:

- There are no metrics or analytics capabilities. Therefore, the *Analytics* page is not included in the product user interface.
- There are no deployment connectors or built-in integrations available. You must build a custom client application that can host the assistant. As a result, the *Integrations* page is not included in the product user interface.
- The @sys-person and @sys-location system entities are not supported and the updated numeric system entities are not available.
- There is no search function in the product.
- You cannot use the Activity Tracker service to track user actions.
- Autocorrection, intent recommendations, and the new irrelevance detection model are not supported.
- The product tour that is available to some first-time users of the cloud-based product is not available.

27 November 2019

IBM Watson® Assistant for IBM Cloud Pak® for Data version 1.4 is available

Watson Assistant for IBM Cloud Pak® for Data version 1.4 is compatible with IBM Cloud Pak® for Data version 2.5.

Czech language not automatically enabled

The Czech language is not enabled automatically anymore.

Assistants and Skills navigation menu update

The main menu options of **Assistants** and **Skills** have moved from being displayed at the top of the page to being shown as icons in a new navigation pane.

Skills secondary navigation menu update

The tabbed pages for the tools that you use to develop a dialog skill were moved to a secondary navigation bar that is displayed when you open the skill.

Rich response types support

Rich response types are now supported in a dialog node with slots. You can display a list of options for a user to choose from as the prompt for a slot, for example. For more information, see Gathering information with slots.

Improved Entities, Dialog, and Intents page responsiveness

The Entities, Dialog, and Intents pages were updated to use a new JavaScript library that increases the page responsiveness. As a result, the look of some graphical user interface elements, such as buttons, changed slightly, but the function did not.

Creating contextual entities is easier

The process you use to annotate entity mentions from intent user examples was improved. You can now put the intent page into annotation mode to more easily select and label mentions. For more information, see Adding contextual entities.

Webhook callouts are available

Add webhooks to dialog nodes to make programmatic calls to an external application as part of the conversational flow. This capability is being introduced as a beta feature. For more details, see Making a programmatic call from dialog.

Testing improvement

You can now see the top three intents that were recognized in a test user input from the **Try it out** pane.

3 September 2019

IBM Watson® Assistant for IBM Cloud Pak® for Data version 1.3 is available

Watson Assistant for IBM Cloud Pak® for Data version 1.3 is compatible with IBM Cloud Pak® for Data versions 2.1.0.1 and 2.1.0.2. There is now added support for installing IBM Cloud Pak for Data with Red Hat OpenShift.

Federal Information Security Management Act support

Federal Information Security Management Act (FISMA) support is available for Watson Assistant for IBM Cloud Pak for Data for this version (V1.3). FISMA support is also available to those who purchased V1.2 (28 June 2019) and upgraded to V1.3. Watson Assistant for IBM Cloud Pak for Data is FISMA High Ready.

Provision more instances in a single deployment

You can now provision up to 30 instances of Watson Assistant in a single deployment.

Search skill now generally available

The search skill is now generally available.

Features not included

This release does not include the following features, which are currently available for cloud instances:

- Autocorrection, intent recommendations, and webhooks are not supported.
- The product tour that is available to some first time users of the cloud-based product is not available.
- The new JavaScript library that is being used in cloud instances to increase the page responsiveness is not in use.

28 June 2019

IBM Watson® Assistant for IBM Cloud Pak® for Data version 1.2 is available

The Watson Assistant tool now works with IBM Cloud Pak for Data 2.1. It does not work with stand-alone IBM Cloud Private. The following changes were made in this release:

Assistants are now available

An assistant can manage user sessions on your behalf.

Search skill is a new beta feature

You can create a search skill to trigger a search in an external data source that you configure in Watson Discovery.

Introducing dialog skills

Instead of creating a workspace as the container for your training data and dialog, you create a dialog skill.

Intent conflict resolution is available

Prevent your assistant from answering the wrong question by keeping your intents distinct from one another. Intent conflict resolution is now available. It can find intents with overlapping user examples, and gives you a graphical user interface in which to fix them. Nondistinct intents can result in misclassifications of user input.

21 February 2019

IBM Watson® Assistant for IBM® Cloud Private version 1.1 is available

The Watson Assistant tool now works with IBM Cloud Private 3.1.0. It does not work with IBM Cloud Private 2.1.0.3. Watson Assistant for IBM Cloud Private version 1.1 is compatible with IBM Cloud Pak® for Data version 1.2.

Decrease in the number of required Virtual Private CPUs

The number of required Virtual Private CPUs has decreased from its previous number (of 60 VPCs).

Improvements to language support

Language support was improved, which means you do not need as many additional resources when you add support for more languages.

23 November 2018

IBM Watson® Assistant for IBM® Cloud Private version 1.0.1 runs on IBM Cloud Private 2.1.0.3

A revised Helm chart (version 1.0.1) was published, which improves the Helm chart and packaging.

New configuration settings

New configuration settings were added that allow you to specify domain names and IP addresses for the coordinator and proxy nodes of the IBM® Cloud Private cluster. A new checkbox is visible for enabling recommendations; however, do not select it as the feature is not fully supported yet.

Development deployment required resources

The resources required for a development deployment changed for Minio from one 20 GB replica to four 5 GB replicas. This change means you need to create 13 persistent volumes instead of 10 to support the deployment.

5 October 2018

IBM Watson® Assistant for IBM® Cloud Private version 1.0.0.1 runs on IBM Cloud Private 2.1.0.3

A revised Helm chart (version 1.0.0.1) was published, which improves the installation process.

26 September 2018

IBM Watson® Assistant for IBM® Cloud Private 1 is available

IBM Watson® Assistant for IBM® Cloud Private version 1 runs on IBM Cloud Private 2.1.0.3.

Introducing the **Build** tab

The IBM Watson® Assistant tool includes a **Build** tab that offers pre-built intents you can add to your workspace from a content catalog, the ability to define your own intents and entities, and has a graphical user interface you can use to build a dialog. The following key features are also available:

- Dialog: Digression and disambiguation support, nodes with slots, rich responses (including *Connect to human agent*)
- Entities: Contextual entities, system entities for currency, date, number, percentage, and time.
- Intents: Content catalog

Features not included

These features are not available from IBM® Cloud Private, but are available in the public IBM Cloud instance at the time of this release:

- There are no metrics or analytics capabilities. Therefore, the *Improve* tab is not included in the tool.
- There are no deployment connectors or built-in integrations available. You must build a custom client application that can host the assistant. As a result, the *Deploy* tab is not included in the tool.
- You cannot search within the tool.
- The @sys-person and @sys-location system entities are not supported.
- You cannot make programmatic calls to Cloud Functions actions from the dialog.
- No entity synonym recommendations are available.
- No intent conflict detection is available.

Release notes for watsonx Assistant for IBM Software Hub

IBM Software Hub

Use these release notes to learn about the latest updates to watsonx Assistant for IBM Software Hub.

Web chat versions

When you install watsonx Assistant for IBM Software Hub, the latest available version of the web chat integration is included. See the following table for details about the latest available web chat version for each watsonx Assistant for IBM Software Hub release. If your web chat version is not locked, then the web chat integration is upgraded to the latest available version when you upgrade watsonx Assistant for IBM Software Hub.

The following table shows the latest version of the web chat integration that is included with each release of watsonx Assistant for IBM Software Hub. IBM Software Hub supports web chat versions 8.3.2 or later. To customize and change version numbers, see Controlling the web chat version.

watsonx Assistant for IBM Software Hub version	Latest web chat version available
5.2.1	8.11.0
5.2.0	8.7.1
5.1.3	8.7.1
5.1.2	8.6.0
5.1.1	8.5.1
5.1.0	8.3.2

Web chat versions in watsonx Assistant for IBM Software Hub

5.2.1 release, 27 August 2025

New features

Automate the clean up of zombie records by using the CLU scheduler

When you create an assistant, the Conversational Language Understanding (CLU) scheduler logs a corresponding entry in its database. If the entry is not properly removed during a requested assistant deletion, the entry becomes a zombie record. The CLU cleanup scheduler identifies and cleans up the zombie records. For more information, see Configuring CLU cleanup scheduler.

Integrate your watsonx Assistant with your customers' preferred messaging tool

You can now set up your assistant to integrate with WhatsApp (by using Twilio for the integration), Facebook Messenger, and Microsoft Teams. Your customers' experiences are enhanced because they can exchange messages with your assistant in their tool of choice. For more information, see the following integration instructions:

- Integrating with WhatsApp
- Integrating with Facebook Messenger
- Integrating with Microsoft Teams

5.2.0 release, 11 June 2025

New features

Integrate watsonx Assistant with Genesys Audio Connector

You can now integrate Genesys Audio Connector with your assistant to stream conversation audio between the assistant and Genesys Cloud. For more information, see Integrating with Genesys Audio Connector.

Secure watsonx Assistant with webhook authentication

Use webhooks in your assistant to communicate with external systems while ensuring security. Define authentication methods to authorize only trusted sources to trigger webhooks. For more information, see Defining the authentication method for pre-message and post-message webhooks.

5.1.3 release, 30 April 2025

Updates

Support for HTTP proxy configuration

watsonx Assistant now supports HTTP proxy configuration. By configuring cluster HTTP proxy settings in IBM Software Hub, you can ensure that your cluster communicates with external repositories and access the required resources to install and update the software. For more information, see Applying cluster HTTP proxy settings to IBM Software Hub.

Update on the search confidence for conversational search

When you make small modifications to text processing, it impacts the values of retrieval confidence, response confidence, and extractiveness. For more information on the scores, see Conversational search analytics.

5.1.2 release, 26 March 2025

New features

Use LLMs for your watsonx Assistant

Large language models (LLMs) are now available for you to configure to improve the capabilities of your assistants. From the Base LLM section of the new Generative AI page, you can enable LLM features for the existing actions in your assistants to improve the conversation capabilities of your assistants. You can choose the foundation model that best interacts with your assistants for conversational search. For more information, see Supported foundation models for GPU features.

Updates

Evaluate testing message limits You no longer have a limit on the number of tests in a test run to evaluate and analyze your watsonx Assistant's performance but each test can include a maximum of 250 messages. For more information, see Testing limits for the set of data.

5.1.1 release, 27 February 2025

New features

Hide or display the skill output

You can now choose to hide or display the skill output that is stored as an assistant variable. For more information, see Passing values to a subaction.

Integrate your watsonx Assistant with Slack

You can integrate your watsonx Assistant with Slack. Then you can configure the assistant to support certain events so that your assistant can respond to questions that are asked in Slack direct messages or in Slack channels where the assistant is directly mentioned. For more information, see Integrating with Slack.

Evaluate assistant's performance

You can now evaluate the performance of your watsonx Assistant by uploading a comprehensive, relevant collection of utterances and sending the utterances to your assistant in a test run. . For more information, see Evaluating the assistant.

Post-process the conversational search responses

You can now save your conversational search responses in action variables of your watsonx Assistant for post-processing. For more information, see Search for the answer.

Updates

Updated conversational search user query

To improve integration with Elasticsearch, the question mark (?) is no longer automatically added to the user query in conversational search. In rare

cases, this change might alter the conversational search responses.

Limit on number of entries for translation download

You can enable language data files to be downloaded for translation, only if the total number of entries in your assistant is 400000 or less. For more information, see Using multilingual downloads for translation.

Updated LLM models for Conversational search

As granite-13b-chat-v2 model is deprecated, you can now use the updated foundation model granite-3-8b-instruct for Conversational search. For more information, see Mirroring images directly to the private container registry .

5.1.0 release, 11 December 2024

For the list of watsonx Assistant known issues, see Limitations and Known issues in watsonx Assistant .

For a list of new features and fixes, see What's new and changed in watsonx Assistant .

Feature parity for IBM® watsonx™ Assistant

IBM® watsonx™ Assistant can be deployed as a managed software as a service (SaaS) or installed on premises. New features are continuously added to each deployment's release cycle. To make informed decisions that align with your business needs, it is important to stay informed about the features available in each deployment.

Stay up to date with the latest features through the Release notes of watsonx Assistant.

The feature parity covers the following deployments:

- IBM Cloud: for the IBM-managed instances that are provisioned as a service on IBM Cloud.
- IBM Software Hub : for client-managed instances that are provisioned on the IBM Software Hub. It is an on-premises deployment and uses a stationary version of the continuous-delivered and continuous-integrated SaaS deployment for each of its releases into IBM Software Hub.

IBM Software Hub refers to the latest released version on on-premises. For more information about the latest version of Software Hub, refer to Latest version.

Parity by features' capabilities

Features can have their own capabilities, which might have limitations on what deployments support them. In the following sections, you can view the capabilities of each feature and if the deployments support these capabilities.

In the tables, you can find the following indications:

- The green check icon indicates that the deployment supports the capability.
- The green check with a warning icon indicates that the deployment supports the capability, but the capability might have specific implementations that differ from other deployments.
- The red cross icon indicates that the deployment does not support the capability.

Generative AI

Feature	IBM Cloud	IBM Software Hub
Select a base model	✓	✓
Add prompt instructions	✓	✓
Enable or disable conversational search for the base LLM	✓	✓

The support of the Generative AI capabilities on each deployment.

Actions

Feature	IBM Cloud	IBM Software Hub
Add Skill-based action	✓	✓
Add Custom-built action	✓	✓

Table. The support of the Actions capabilities on each deployment.

Global settings

Feature	IBM Cloud	IBM Software Hub
Ask clarifying questions	✓	⚠ 1
Change conversation topic and as confirmation questions	✓	✓
Use watsonx.ai for information gathering	✓	✓
Detect mentions of hate, abuse and profanity (HAP) in the content	✓	✗

Autocorrection for spelling mistakes	✓	✓
Customize formats for local and date and time	✓	✓
Set an algorithm version for the assistant training	✓	✓
Upload and download the assistant's actions	✓	✓

The support of the Global settings capabilities on each deployment.

[1] **IBM Software Hub :** Customizing response modes for new actions is not available.

Evaluate

Feature	IBM Cloud	IBM Software Hub
Evaluate response settings	✓	✓

The support of the Evaluate capabilities on each deployment.

Preview

Feature	IBM Cloud	IBM Software Hub
Preview assistant	✓	✓

The support of the Preview capabilities on each deployment.

Publish

Feature	IBM Cloud	IBM Software Hub
Publish content and revert publishing	✓	✓
View all versions	✓	✓
Download published version	✓	✓

The support of the Publish capabilities on each deployment.

Environments

Feature	IBM Cloud	IBM Software Hub
Add and preview environment	✓	✓
Get API details for the environment	✓	✓
Set up pre-message webhook	✓	✓
Set up post-message webhook	✓	✓
Set up log webhook	✓	✓
Set up audio webhook	✓	✓
Set up inactivity history	✓	✓
Enable session history	✓	✓
View the added channels, content and extensions for the environment	✓	✓

The support of the Environments capabilities on each deployment.

Analyze

Feature	IBM Cloud	IBM Software Hub
View data for action completion	✓	✓
View data for conversational search responses	✓	✓
View data for recognized and unrecognized requests	✓	✗
View data about conversations with assistants	✓	✓

The support of the Analyze capabilities on each deployment.

Integration with web chat

Feature	IBM Cloud	IBM Software Hub
Customize the chat UI	✓	✓
Edit the desktop and mobile launcher	✓	✓
Edit the chat home screen	✓	✓
Add live chat integration with eGain provider	✓	✓
Add live chat integration with Enghouse provider	✓	✓
Add live chat integration with Genesys SIP provider	✓	✓
Add live chat integration with Kustomer provider	✓	✓
Add live chat integration with NICE inContact provider	✓	✓
Add live chat integration with Oracle provider	✓	✓
Add live chat integration with Salesforce provider	✓	✗
Add live chat integration with Twilio provider	✓	✓
Add live chat integration with Zendesk provider	✓	✗
Add live chat integration with your own provider	✓	✓
Add suggestion to connect to agent	✓	✓
Enable web chat security	✓	✓
Get the code to embed the web chat	✓	✓

The support of the Integration with Web chat capabilities on each deployment.

Integration with Phone

Feature	IBM Cloud	IBM Software Hub
Connect to Genesys SIP provider	✓	✗

Connect to Genesys Audio Connector provider	✓	✓ [1]
Connect to Twilio provider	✓	✗
Connect to NICE inContact provider	✓	✗
Bring your own phone provider	✓	✗
Manage phone numbers	✓	✗
Configure speech to text and text to Speech	✓	✗
Configure SIP trunking security options	✓	✗
Handle call and transfer failures	✓	✗

The support of the Integration with Phone capabilities on each deployment.

[1] **IBM Software Hub :** Connecting to Genesys Audio Connector and speech configuration might be available via patch.

Integration with other channels

Feature	IBM Cloud	IBM Software Hub
Add SMS as a channel and customize provider with IntelePeer and Twilio	✓	✗
Add and customize Facebook messenger as a channel	✓	✓
Add and customize Genesys Bot Connector as a channel	✓	✓ [1]
Add and customize Slack as a channel	✓	✓
Add and customize Microsoft teams as a channel	✓	✓
Add and customize WhatsApp with Twilio as a channel	✓	✓

The support of the Integration with other channels capabilities on each deployment.

[1] **IBM Software Hub :** Genesys Bot Connector as a channel might be available via patch.

Search integration

Feature	IBM Cloud	IBM Software Hub
Add search integration with Elasticsearch	✓	✓
Add search integration with Milvus	✓	✗
Add search integration with a custom service	✓	✓
Configure contextual awareness for conversational search	✓	✓
Configure the search for conversational search	✓	✓
Configure citations for conversational search	✓	✓
Configure the content for "No results" and "Connectivity issues"	✓	✓

The support of the Search integration capabilities on each deployment.

Integration with other extensions

Feature	IBM Cloud	IBM Software Hub
Add integration with Segment as an extension	✓	✗
Build a custom extension	✓	✓

The support of the Integration with other extensions capabilities on each deployment.

Activity Log

Feature	IBM Cloud	IBM Software Hub
View the activity history for the assistant	✓	✓

The support of the Activity Log capabilities on each deployment.

Dialog support

Feature	IBM Cloud	IBM Software Hub
Adding dialog	✓	✓
Dialog creation workflow	✓	✓
Creating intents	✓	✓
Using built-in intents	✓	✓
Creating a dialog	✓	✓
Testing your dialog	✓	✓
Gathering information with slots	✓	✓
Managing workflow with versions	✓	✓

Dialog support on each deployment.

Assistant settings

Feature	IBM Cloud	IBM Software Hub
View the assistant details	✓	✓
Configure security certificates (SSL/TLS)	✓	✓
Get the details of assistant IDs and API data	✓	✓
Download or upload assistant data	✓	✓
Delete assistant	✓	✓

The support of the Assistant settings capabilities on each deployment.

Use NeuralSeek to return polished answers from existing help content

In this tutorial, you will use the Watson Discovery, watsonx Assistant, and NeuralSeek services that are available from the IBM Cloud catalog to create a virtual assistant that can answer questions about Watson Discovery. The assistant will generate answers by using the existing Watson Discovery product documentation as its knowledge base.

Note: This tutorial shows the steps for creating a managed deployment of Discovery. However, you can create a Discovery service instance that is either hosted by IBM Cloud or installed in IBM Cloud Pak for Data and connect it to a NeuralSeek service instance.

Learning objectives

By the time you finish the tutorial, you will understand how to:

- Create a Document Retrieval project in Discovery.
- Upload PDF documents to your project, and apply a user-trained Smart Document Understanding model to your PDFs.
- Connect your Discovery project to a NeuralSeek service instance. NeuralSeek is an AI-powered answer generation engine.
- Create an assistant in watsonx Assistant and apply a NeuralSeek integration to it.
- Add an action to your watsonx Assistant that connects to NeuralSeek for answers.
- Use your assistant to answer questions about Discovery.

Important: NeuralSeek is a third-party product that is provided by a vendor outside of IBM and is subject to a separate agreement between you and the third party, if you accept their terms. IBM is not responsible for the product and makes no privacy, security, performance, support, or other commitments regarding the product.

Duration

This tutorial will take approximately 4 to 5 hours to complete.

Prerequisite

1. Before you begin, you must set up a paid account with IBM Cloud.

   You can complete this tutorial at no cost by using a Plus plan, which offers a 30-day trial at no cost. However, to create a Plus plan instance of the service, you must have a paid account (where you provide credit card details). For more information about creating a paid account, see Upgrading your account.

2. Create a Plus plan Discovery service instance.

   Go to the Discovery resource page in the IBM Cloud catalog and create a Plus plan service instance.

   Specify **Dallas** as the location.

   As part of this tutorial, you will provision other services also. The services must be hosted in the same data location so that they can connect to one another. Because the NeuralSeek service is available only from Dallas, you will create all of the service instances in Dallas.

Important: If you decide to stop using the Plus plan and don't want to pay for it, delete the Plus plan service instance before the 30-day trial period ends.

Step 1: Get the product documentation

To use the Discovery product documentation as our knowledge base, we will download the product documentation as a PDF file.

1. From a web browser, go to the product documentation site.

   ```
   https://cloud.ibm.com/docs/discovery-data
   ```

2. From the table of contents panel, click the overflow menu icon in the *Product guide* section, and then choose **View as PDF**.

3. Save the PDF file to your system by clicking the *Save* icon from the page header.

4. Use a PDF file editor to split the PDF document into two separate PDF files of similar size.

Splitting the PDF creates two smaller files that can be enriched faster in Discovery.

Step 2: Create a Document Retrieval project

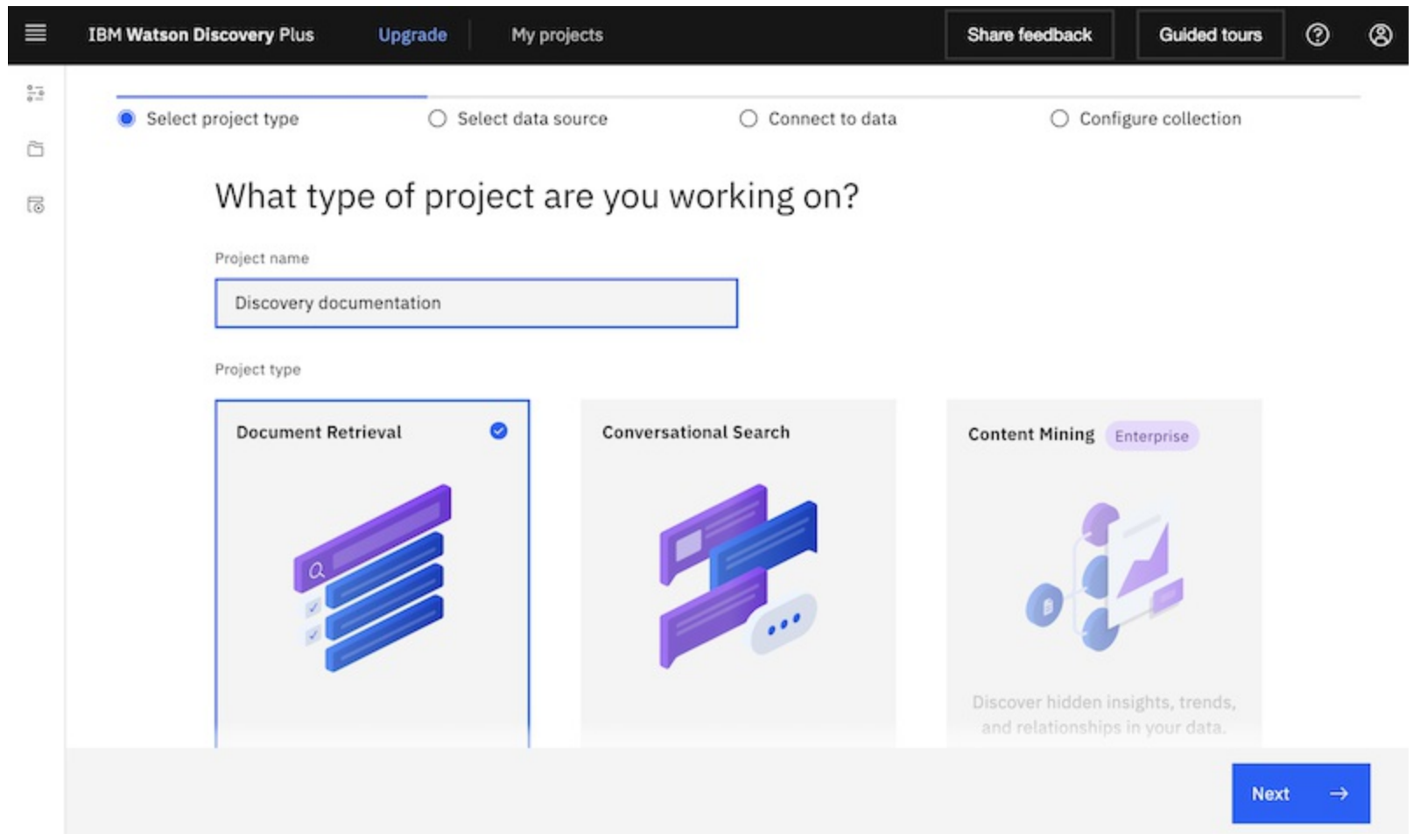
Now that you have the latest copy of the product documentation, add it to a Discovery project as your data source.

In Discovery, you will create a *Document Retrieval* project type. Documents that you add to a project of this type are automatically enriched in the following ways:

- Entities, such as proper nouns, are identified and tagged.
- Parts of speech are identified and tagged.

This tagged information is used later when a natural language phrase is submitted as a search query to return an accurate response.

1. Open a new web browser page.
2. From the Discovery Plus plan service page in IBM Cloud, click **Launch Discovery**.
3. From the *My Projects* page, click **New Project**.
4. Name your project `Discovery documentation`, and then click the **Document Retrieval** tile.



Project type options

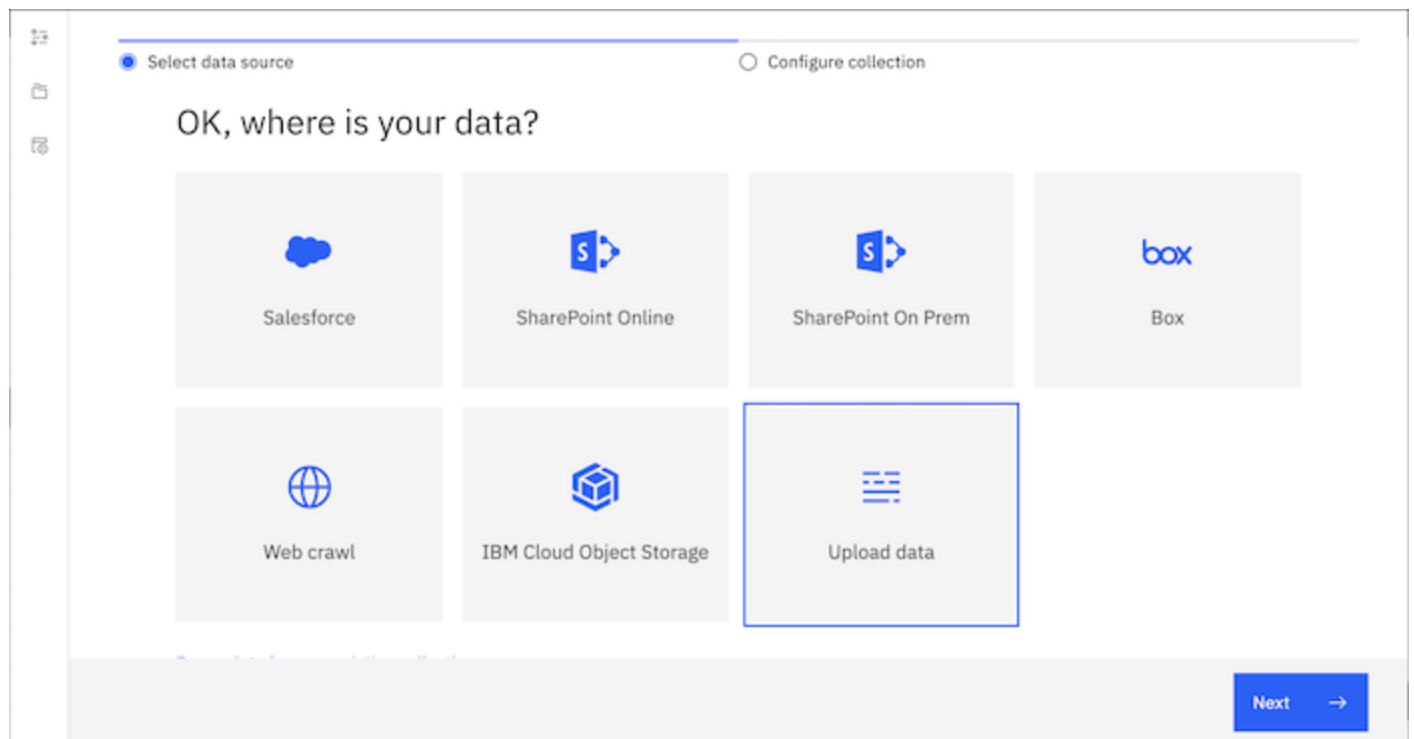
5. Click **Next**.

You'll configure the data source for the project in the next step.

Step 3: Upload data to the project

Add the documentation PDFs to your Discovery project.

1. From the *Select data source* page, click the **Upload data** tile, and then click **Next**.



Creating a collection from uploaded data

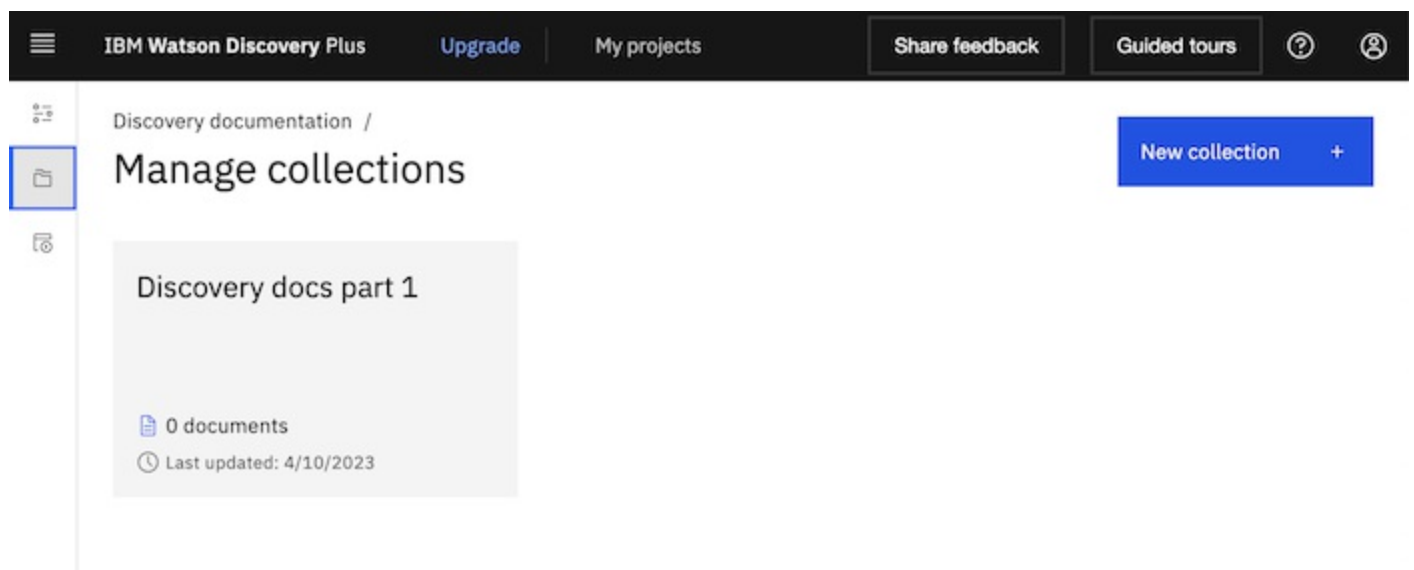
2. Name the collection *Discovery docs part 1*, and then click **Next**.

3. Click **Drag and drop files here or upload**, and then browse to add the first PDF file that you created earlier.

4. Click **Finish**.

   Your file is processed as it is added to the collection.

5. From the navigation panel, click **Manage collections**, and then click **New collection**.



Adding a second collection

6. Repeat the previous steps to add the second PDF file as a collection named *Discovery docs part 2*.

After the data is uploaded, it is processed and indexed by Discovery. While the data is being processed, let's create our virtual assistant.

Step 4: Create an assistant

For this tutorial, you will create an assistant with a single action. First, you must create a watsonx Assistant service instance.

Both Lite and Trial plan watsonx Assistant service instances are available at no cost. You will create a Trial plan.

1. From a new web browser tab, return to the IBM Cloud catalog.

   Tip: Keep the Discovery page open in a separate tab, so you can switch between the two applications.

2. From the watsonx Assistant resource page in the IBM Cloud catalog, create a Trial plan watsonx Assistant service instance in the Dallas location.

3. From the watsonx Assistant plan service page in IBM Cloud, click **Launch watsonx Assistant**.

   The watsonx Assistant product user interface is displayed where you can create your first assistant.

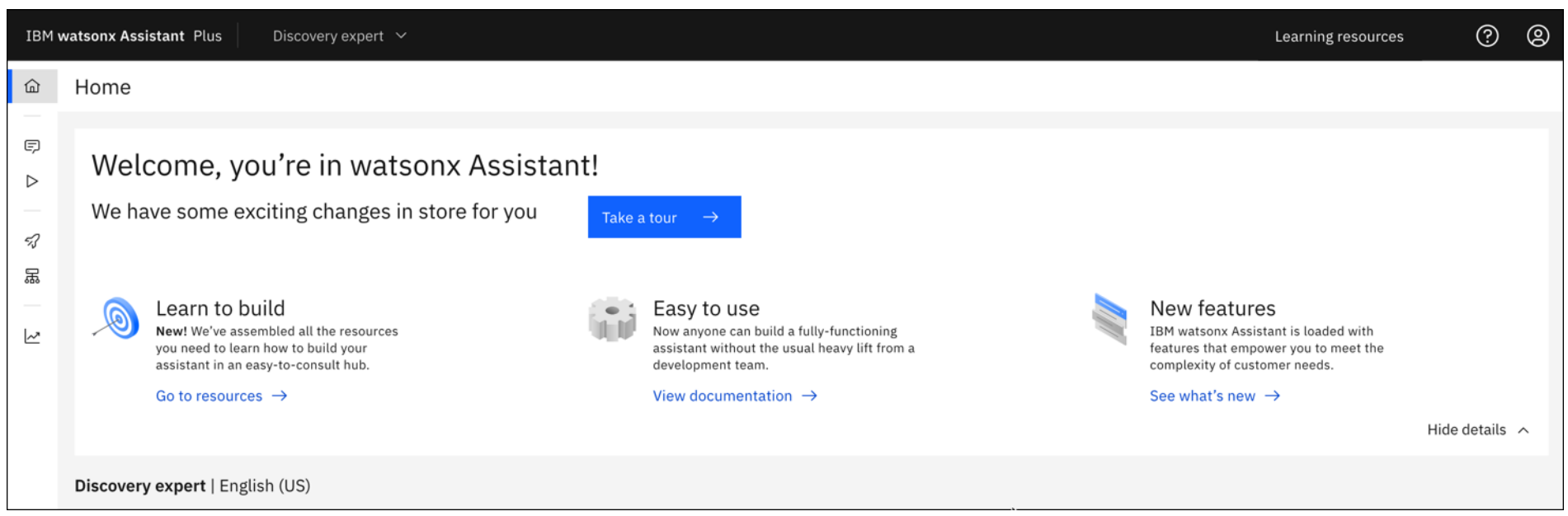
4. Add `Discovery expert` as the assistant name, and then click **Next**.

5. If you are asked to share information about you and your assistant, complete the required fields, and then click **Next**.

   When you create an assistant, a web chat application is created for you automatically.

6. Click **Create** to create the assistant and the corresponding web chat app.

After a congratulatory message, the home page for your new assistant is displayed.



Assistant home page

Before we add anything to our new assistant, let's check on the status of our data.

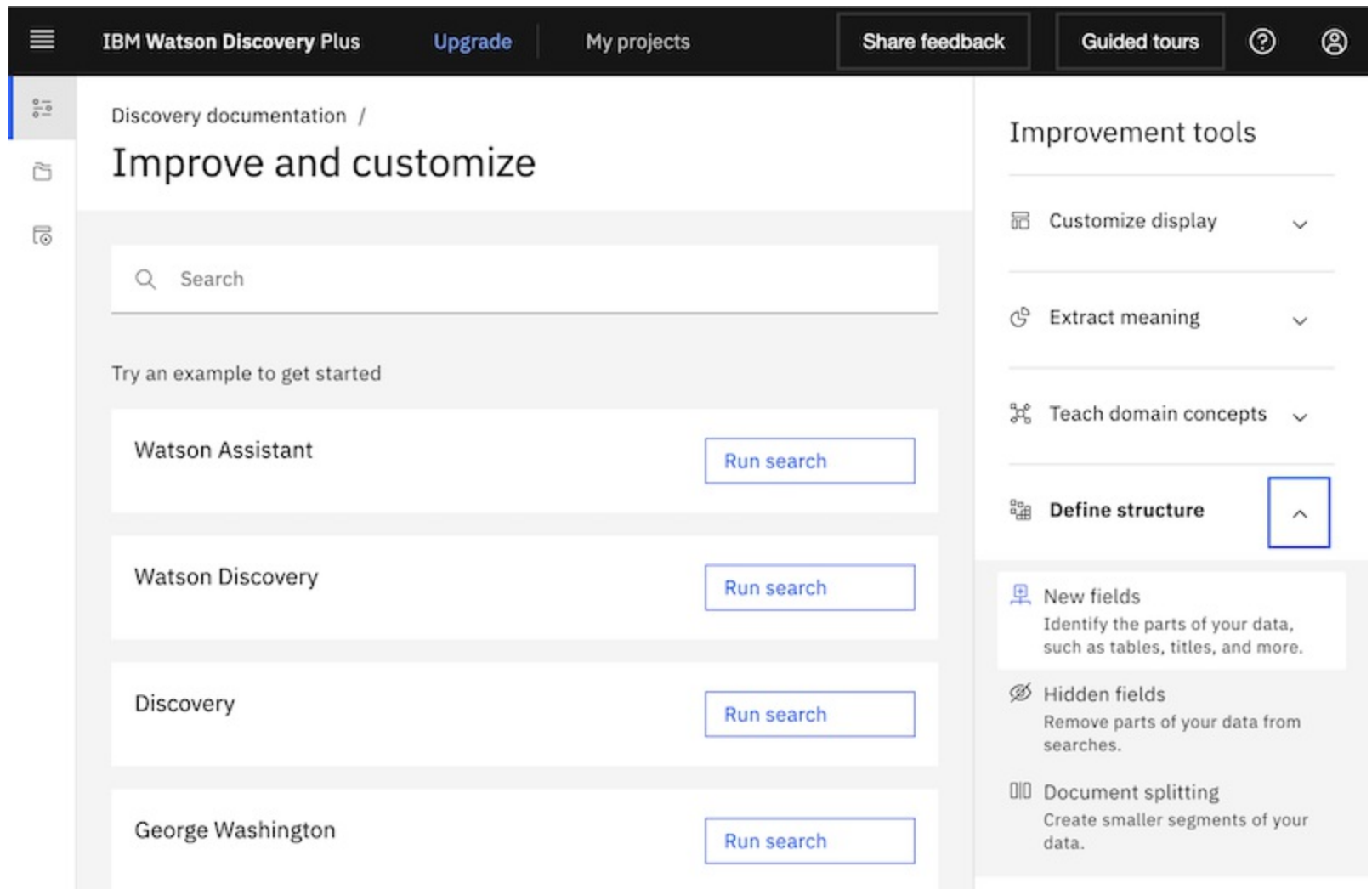
Step 5: Prepare your data for retrieval

To improve the retrievability of the information in your PDF files, you will split the PDF files into many smaller documents. To do so, you will first teach Discovery about the structure of your PDF files, so it understands how subsections are formatted and can split the document by subsection.

1. Return to the web browser tab where your Discovery project is displayed.

   The *Improve and customize* page for the last PDF file that you uploaded is displayed.

2. From the *Improvement tools* panel, expand *Define structure*, and then click **New fields**.

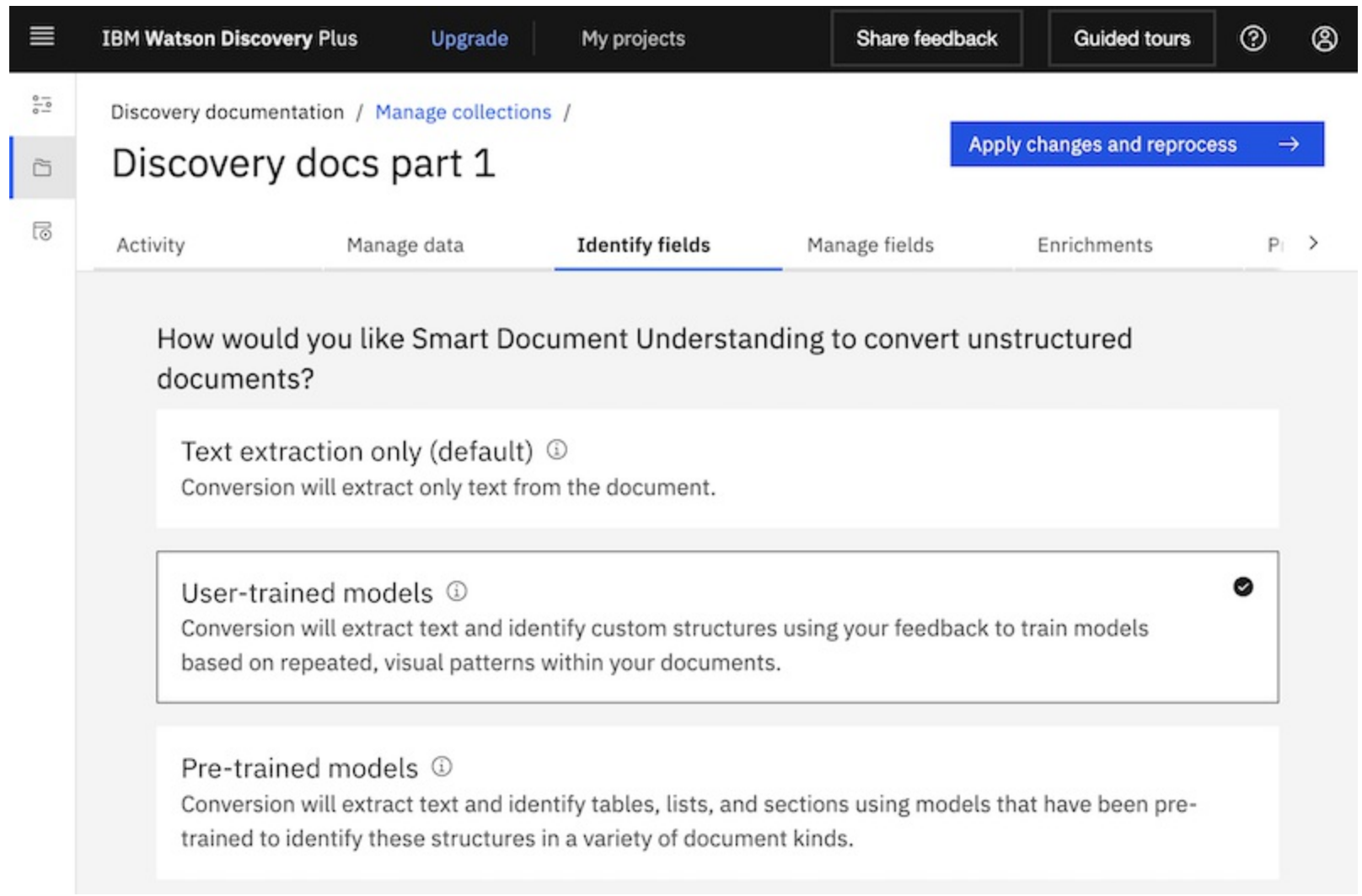


Opening the tool for defining fields

3. Choose the *Discovery docs part 1* collection.

   The Identify fields tab is displayed, where you can choose the type of Smart Document Understanding model that you want to use.

4. Click **User-trained models**, and then click **Submit**.



Creating a user-trained model

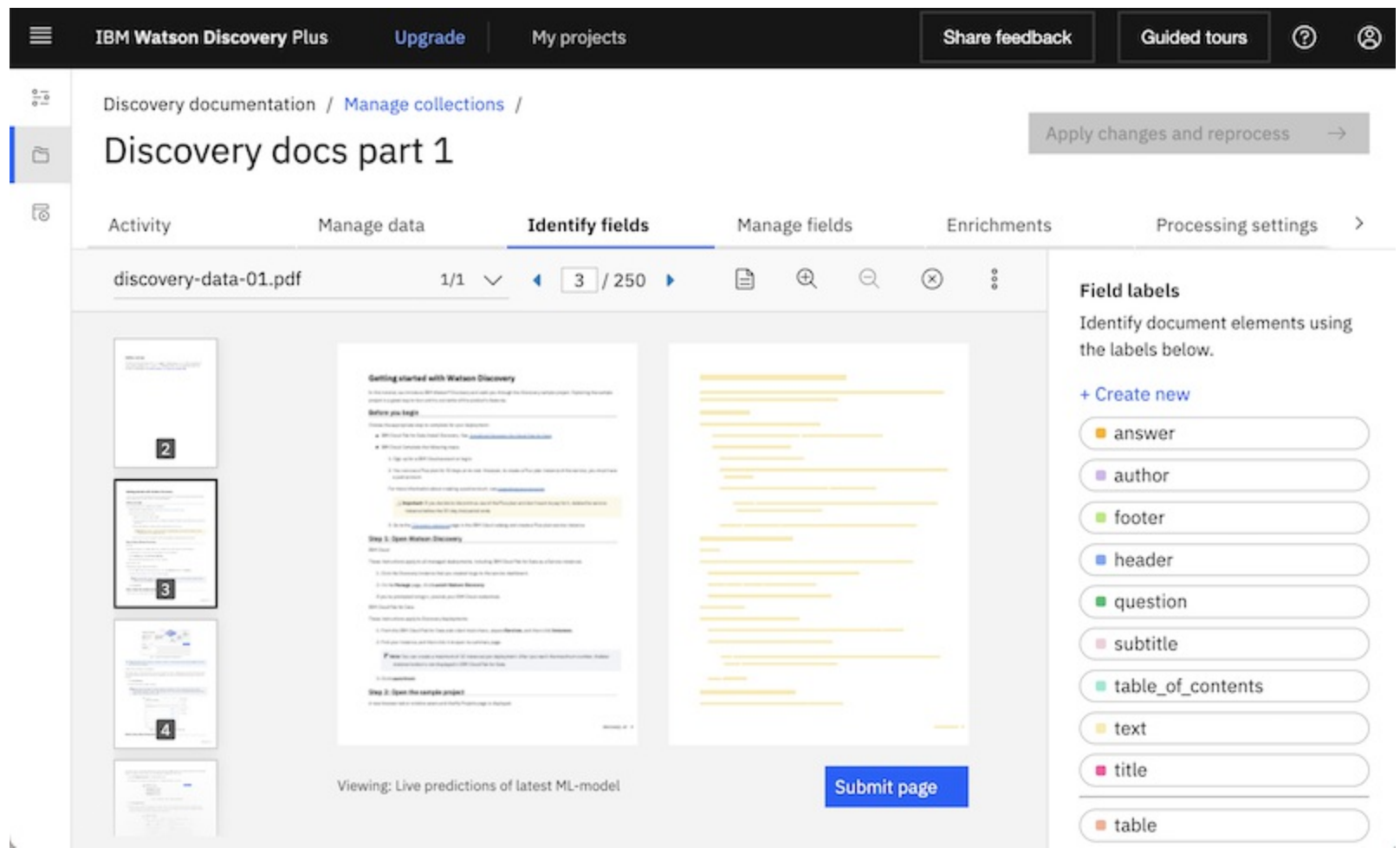
5. Click **Apply changes and reprocess**.

   After some processing occurs, a representation of the document is displayed in the Smart Document Understanding tool. The tool shows you a view of the original document along with a representation of the document, where the text is replaced by blocks. The blocks represent field types.

   Initially, the blocks are labeled as `text` because all of the document content is considered to be standard text by default, and is indexed in the `text` field.

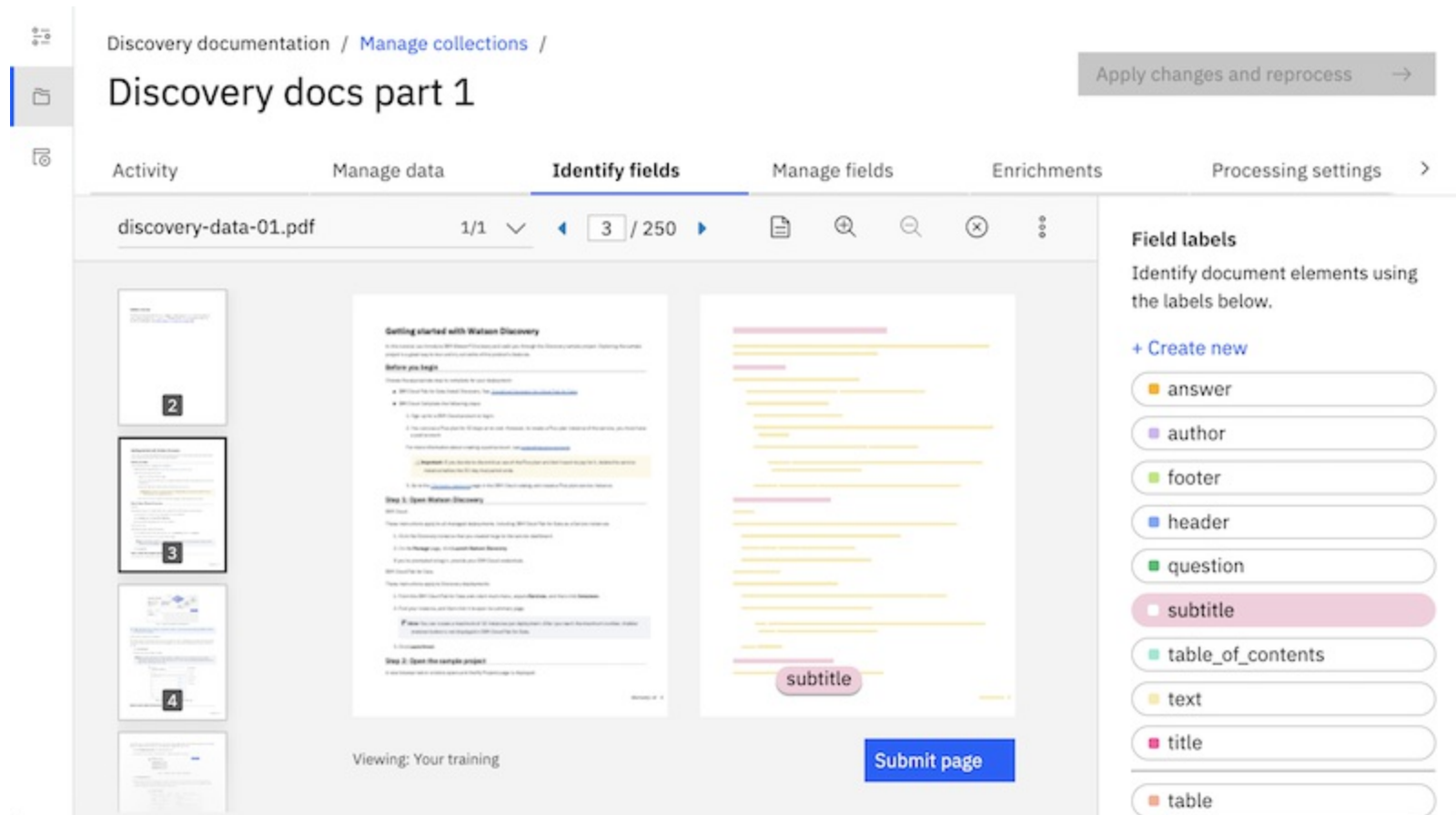
   We want to label all first- and second-level headings as subtitles instead of text.

6. From the thumbnails view, click the thumbnail for the first full-text page from the document to open the first page with real content.



The Smart Document Understanding tool

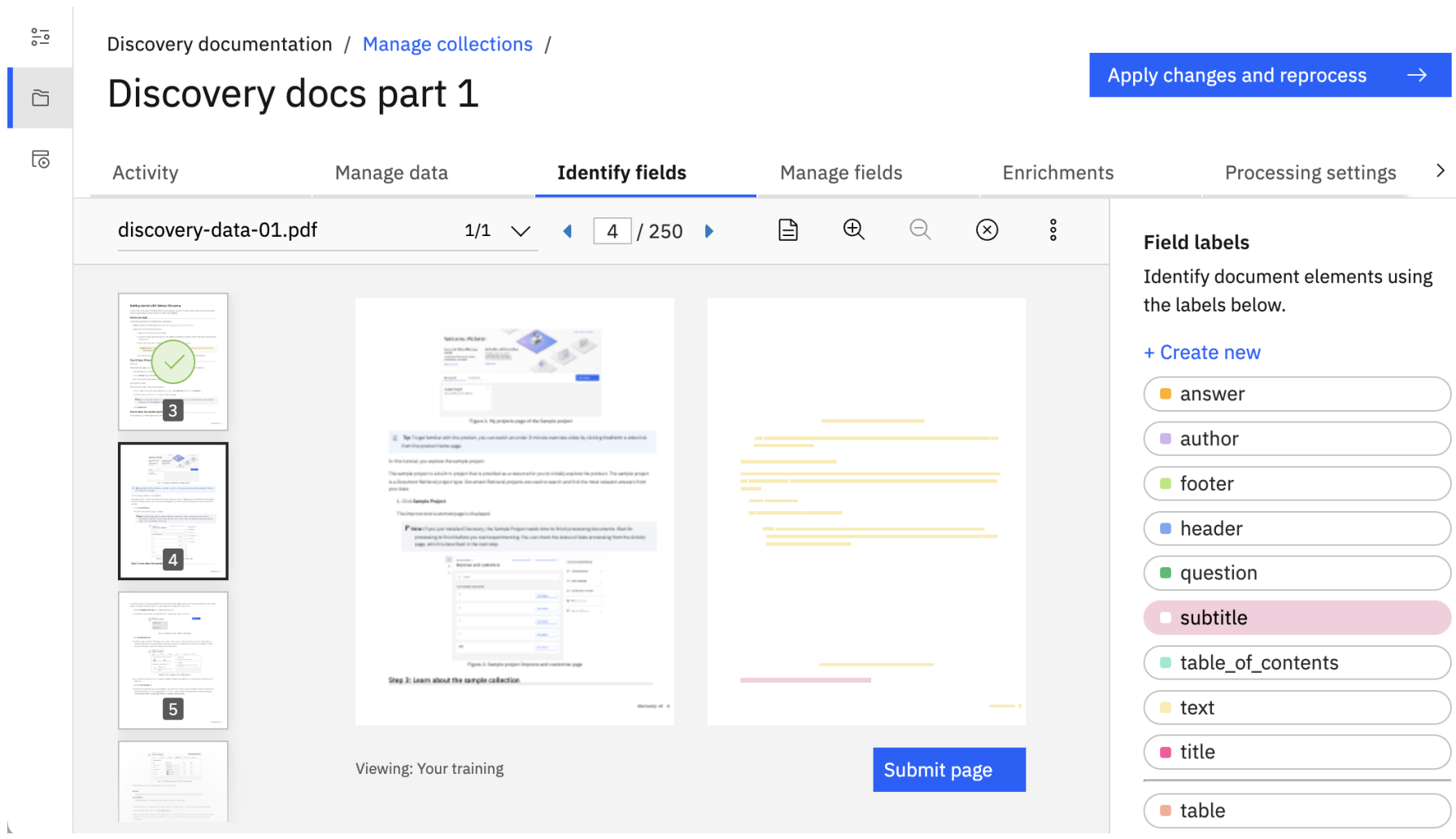
7. To annotate the document, click the `subtitle` label from the *Field labels* list. Then, click each block in the representation of the PDF page that represents a heading to change its label from `text` to `subtitle`.



Applying the subtitle label

8. After every subtitle on the current page is labeled, click **Submit page**.

   The next page of the PDF file is displayed.



Next page is displayed for labeling

9. Repeat this process until the tool is able to label the headings correctly for you in a consistent way when new pages are loaded into the tool. At that point, click **Apply changes and reprocess**.

   Congratulations! You have successfully trained a Smart Document Understanding (SDU) model that can recognize subtitles in your documents. Let's apply the same model to the other PDF file that you added to the project.

10. From the SDU editor toolbar, click the overflow menu icon ⋮ from the page header, and choose **Export model**.
11. Save the `.sdumodel` file to your system in a location where you can access it again shortly.
12. From the navigation panel, click **Manage collections**, and then open the *Discovery docs part 2* collection.
13. Open the *Identify fields* tab.
14. Click **User-trained models**, and then click **Submit**.
15. Click **Apply changes and reprocess**.

16. From the SDU editor toolbar, click the overflow menu icon ⋮, and choose **Import model**, and then click **Select model**.
17. Browse to find the `.sdumodel` file that you downloaded earlier, and then click **Open**.
18. Click **Apply changes and reprocess** to apply the same SDU model to the first collection.

Discovery reprocesses the data in its index to identify subtitles in the documents. While the data is being reprocessed, let's create our answer generator.

Step 6: Create a NeuralSeek service instance

You can use a search extension in watsonx Assistant to connect your assistant directly to Discovery and return passages straight from the data source. However, we will add the NeuralSeek service between watsonx Assistant and Discovery in this tutorial. NeuralSeek retrieves the passages from Discovery and then converts them into answers that sound more conversational.

1. From a new web browser tab, return to the IBM Cloud catalog.

   > **Tip:** Keep the pages to the other services open in separate tabs, so you can switch between the different service instances.

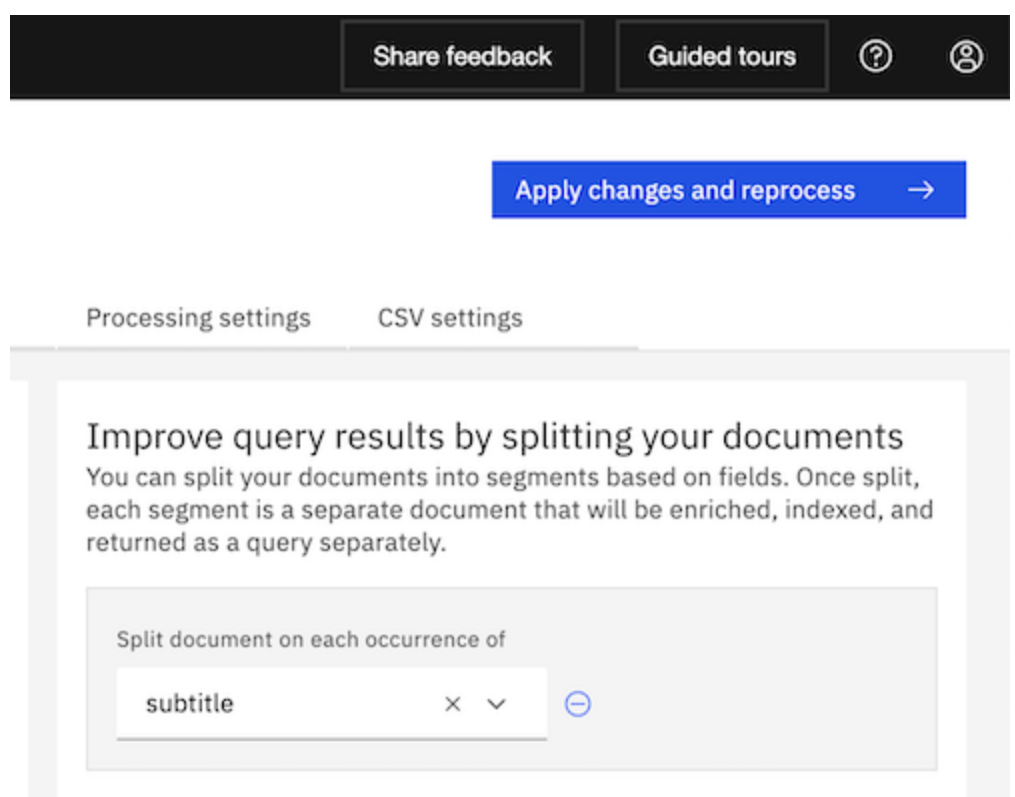
2. From the NeuralSeek resource page in the IBM Cloud catalog, create a Lite plan service instance.
3. On the *Configure* page, add details about your Discovery service instance and customize the connection.
   - You can get the service URL and API key from the Discovery service instance details page in IBM Cloud.
   - The project ID is available from the IBM Cloud user interface. To get it, click **Integrate and deploy** from the navigation panel. Open the *API Information* page, and then copy the project ID.
   - Set the document score range to 50%.
   - Change the snippet character size to 400.
   - Specify your company as the company display name.
   - Change the minimum confidence percentage to 50.

   Click **Save**.

Step 7: Split your PDF documents

Now that subtitles are indexed properly in Discovery, use them as the basis for splitting the PDF files into many smaller documents.

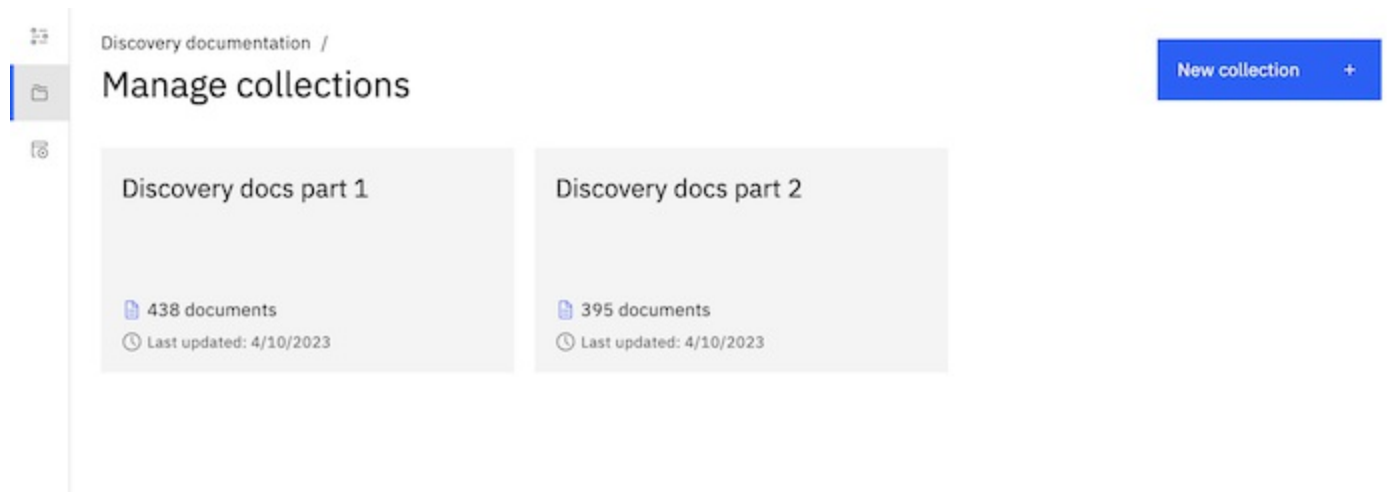
1. Return to the web browser tab where your Discovery project is displayed.
2. Open the **Manage fields** tab for the current collection.
3. In the *Split document on each occurrence of* field, choose **subtitle**, and then click **Apply changes and reprocess**.



Split a document

4. From the navigation panel, click **Manage collections**, and then open the other collection.
5. Go to the *Manage fields* page, and then choose **subtitle** in the *Split document on each occurrence of* field.
6. Click **Apply changes and reprocess**.

The collections start to be reprocessed. After reindexing is finished, instead of containing one document each, the collections will contain several hundred documents each.



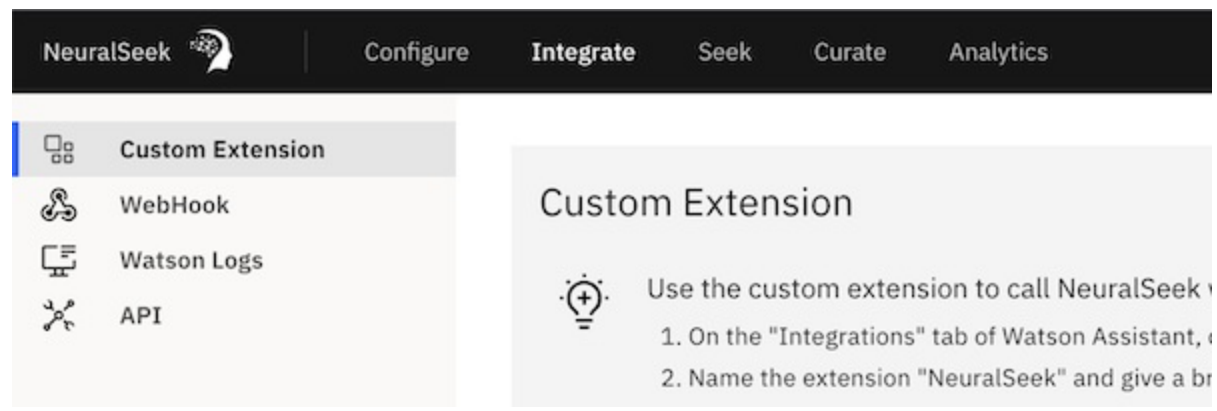
The collections with more documents

While the index is being rebuilt, let's get our assistant ready.

Step 8: Add an extension to your assistant

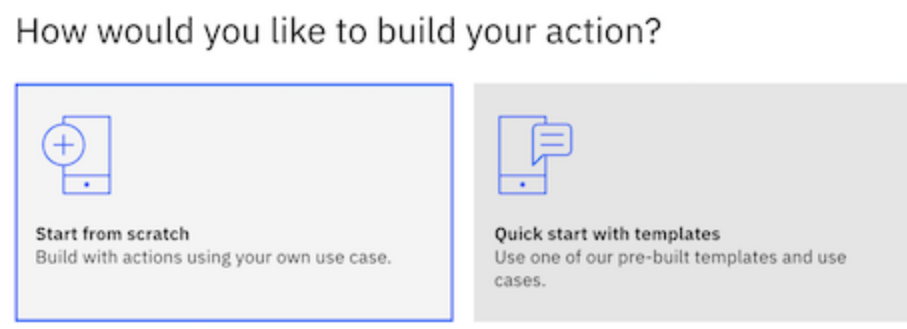
Connect your assistant to your NeuralSeek service instance.

1. Reopen the NeuralSeek service from IBM Cloud. You can find the instance in the *AI and Machine Learning* section of your resource list.
2. Click the *Integrate* tab and follow the instructions to set up the NeuralSeek custom extension for your assistant. Return to this procedure when you're ready to create the action.



Set up the NeuralSeek instance integration

3. From the watsonx Assistant navigation panel, click **Actions**, and then click **New action**.
4. Choose **Quick start from templates**, and then scroll to find and click the **NeuralSeek Starter kit**.



Choose Quick start from template

5. Click **Select this starter kit**, and then click **Add templates**.
6. Click to open the *NeuralSeek search* action that you just added to the assistant.
7. Add the following user example queries to the first step in the action:

What Watson Discovery project types are available and what do they do?

What external data sources are supported by Watson Discovery?

Can I add a custom dictionary to Watson Discovery?

How do I use the Content Mining application?

When should I add query expansions to my project?

Which file types support Smart Document Understanding models?

Can I enable optical character recognition for all file types?

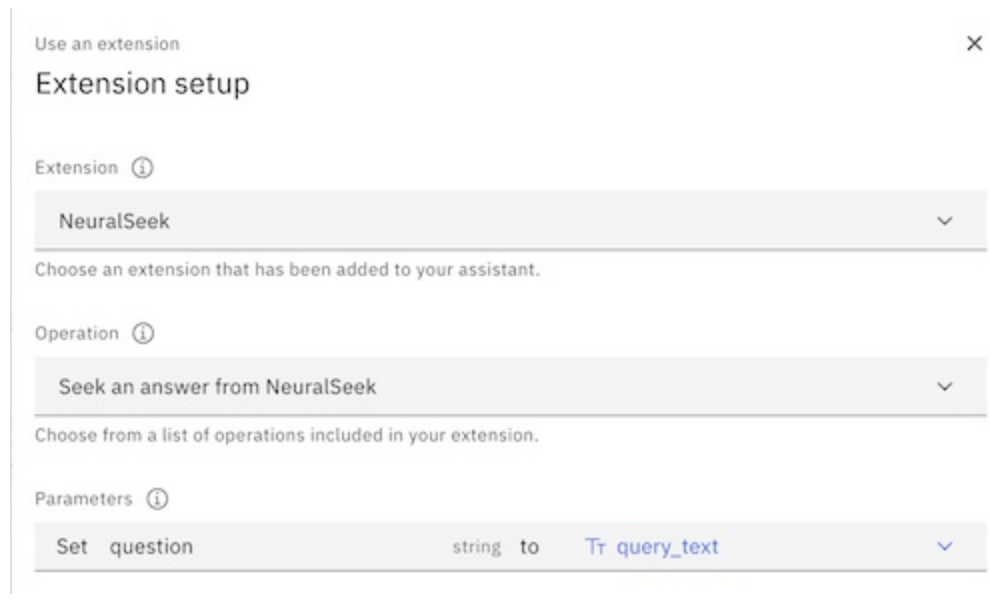
Does my data have to be written in English?

watsonx Assistant uses the sample questions to recognize the types of user questions it should route to this action.

8. Click to open Step 3 for editing.

   In the *And then* section, click **Edit extension**.

   Choose **NeuralSeek**, and then click **Apply**.



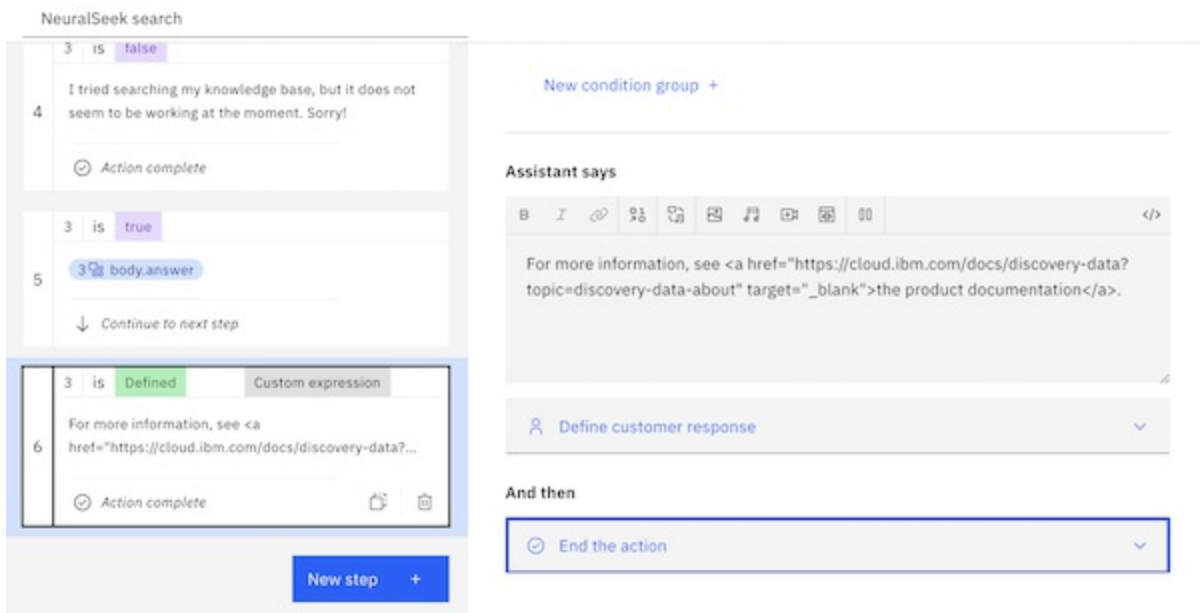
Set up the NeuralSeek extension

9. Click to open Step 6 for editing.

   This step shows a link that users can click to get more information. We want this link to go directly to the product documentation on the IBM Cloud Docs site.

   Change the hypertext reference in the anchor HTML element to contain the following URL:

```
<a href="https://cloud.ibm.com/docs/discovery-data?topic=discovery-data-about" target="_blank">
```



{: caption="Change the URL for the *More information* link" caption-side="bottom"}

10. Save your changes, and then click the X to close the step.

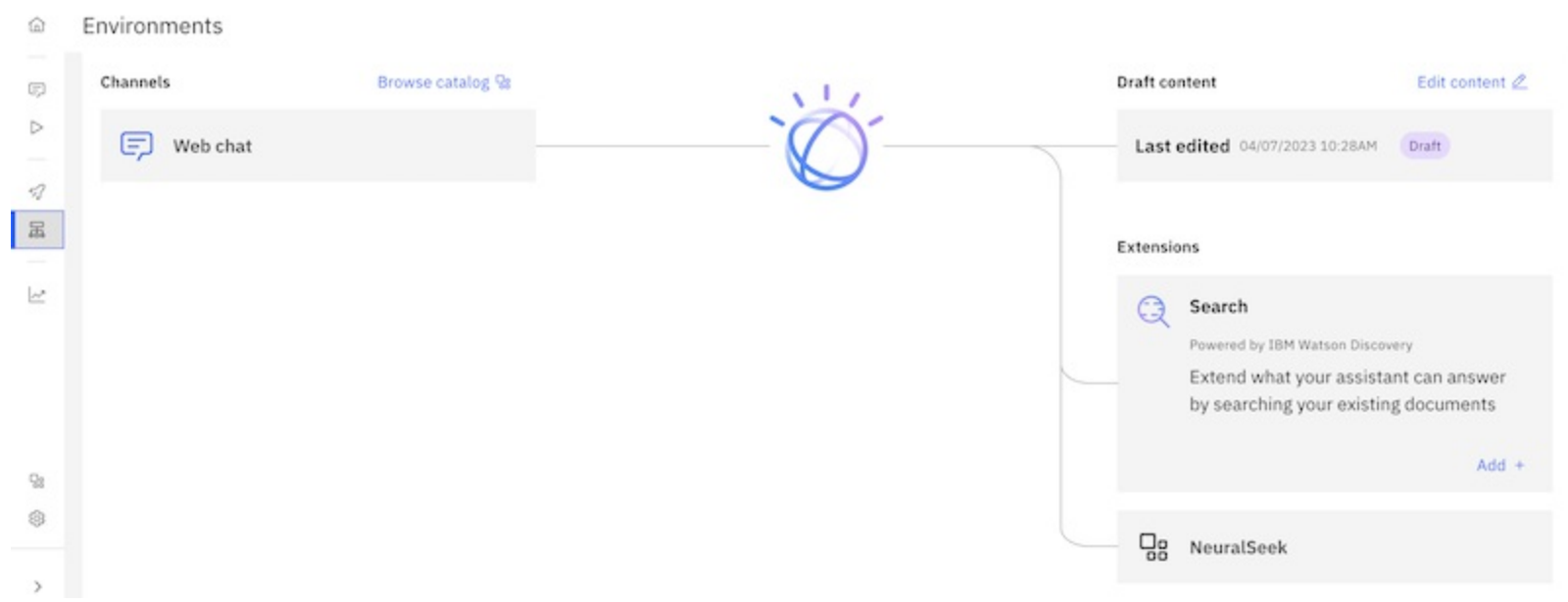
Congratulations! You successfully created an action that recognizes questions about Discovery, and gets its answers from the connected NeuralSeek extension.

Step 9: Configure the web chat for your assistant

To preview your assistant, you will use the built-in web chat as the chat user interface for interacting with the assistant.

1. From the navigation panel in watsonx Assistant, click **Environments**.

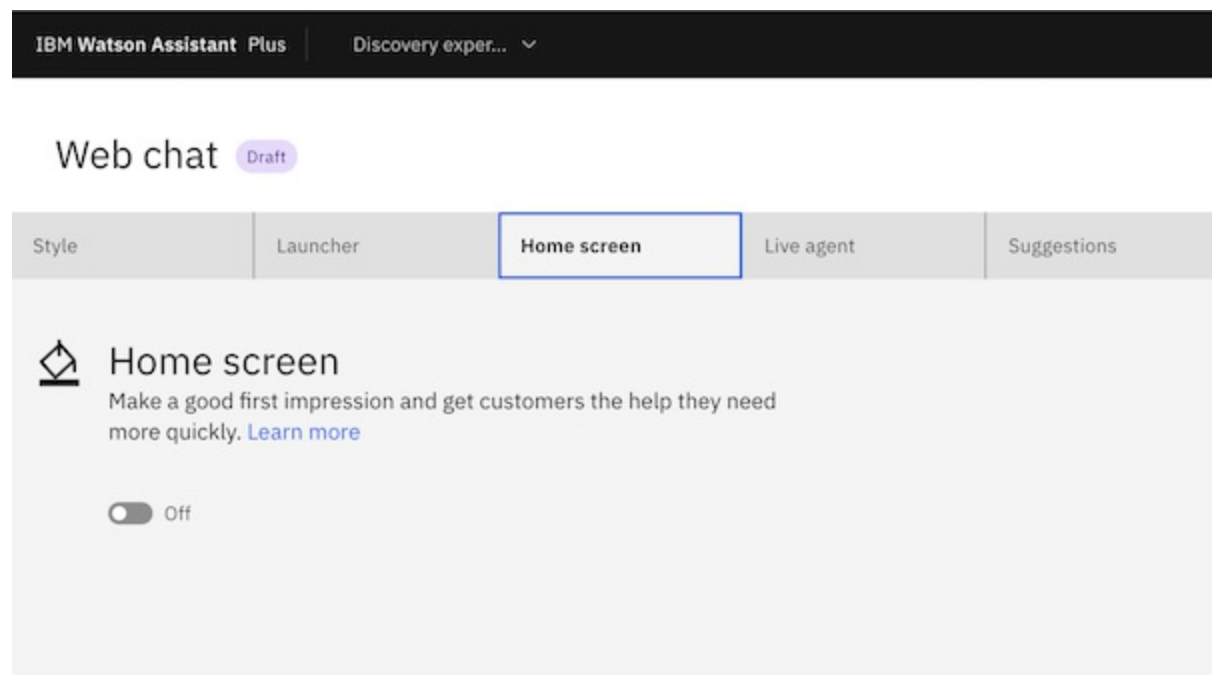
The draft environment is displayed. It shows that a web chat is connected to your assistant. You also can see that the web chat is connected to the NeuralSeek extension.



Environment diagram

2. Click the *Web chat* tile to edit the web chat.

   We don't want to add multiple starter questions, so we're going to turn off the home screen for the web chat. Click the **Home screen** tab. Set the switcher to **Off**, and then click **Save and exit**.



Web chat home screen disabled

You're ready to preview your assistant!

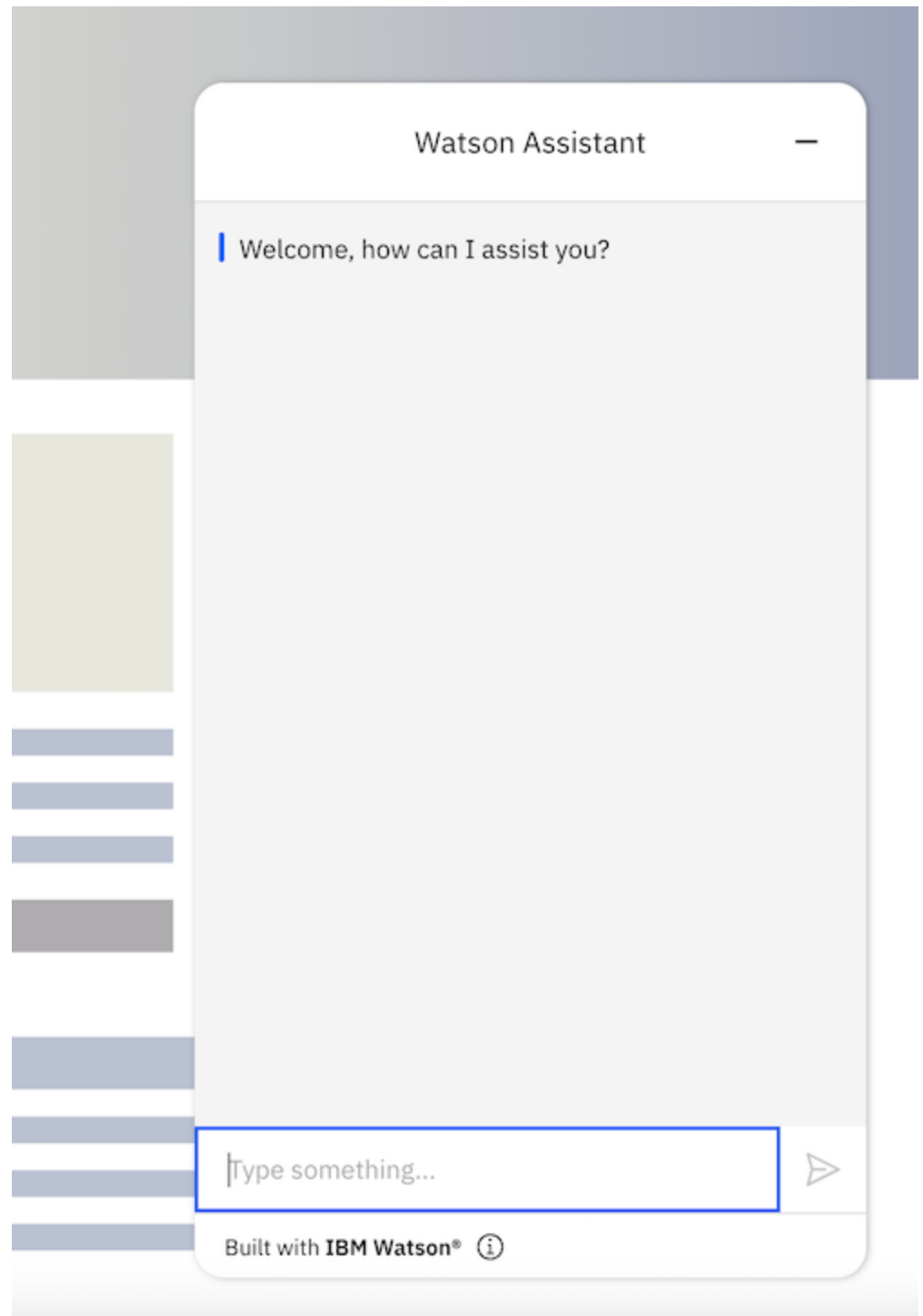
Step 10: Preview the assistant

To preview an assistant that connects to data that is stored in Discovery, you must preview the assistant from the *Environments* page. When you preview the web chat independently, the assistant is not able to retrieve data from Discovery; it needs the environment resources to be able to connect to Discovery.

1. From the *Environments* page, click **Preview this environment**.

   A sample web page is displayed that includes a chat icon .

2. Click the chat icon to open the web chat window.

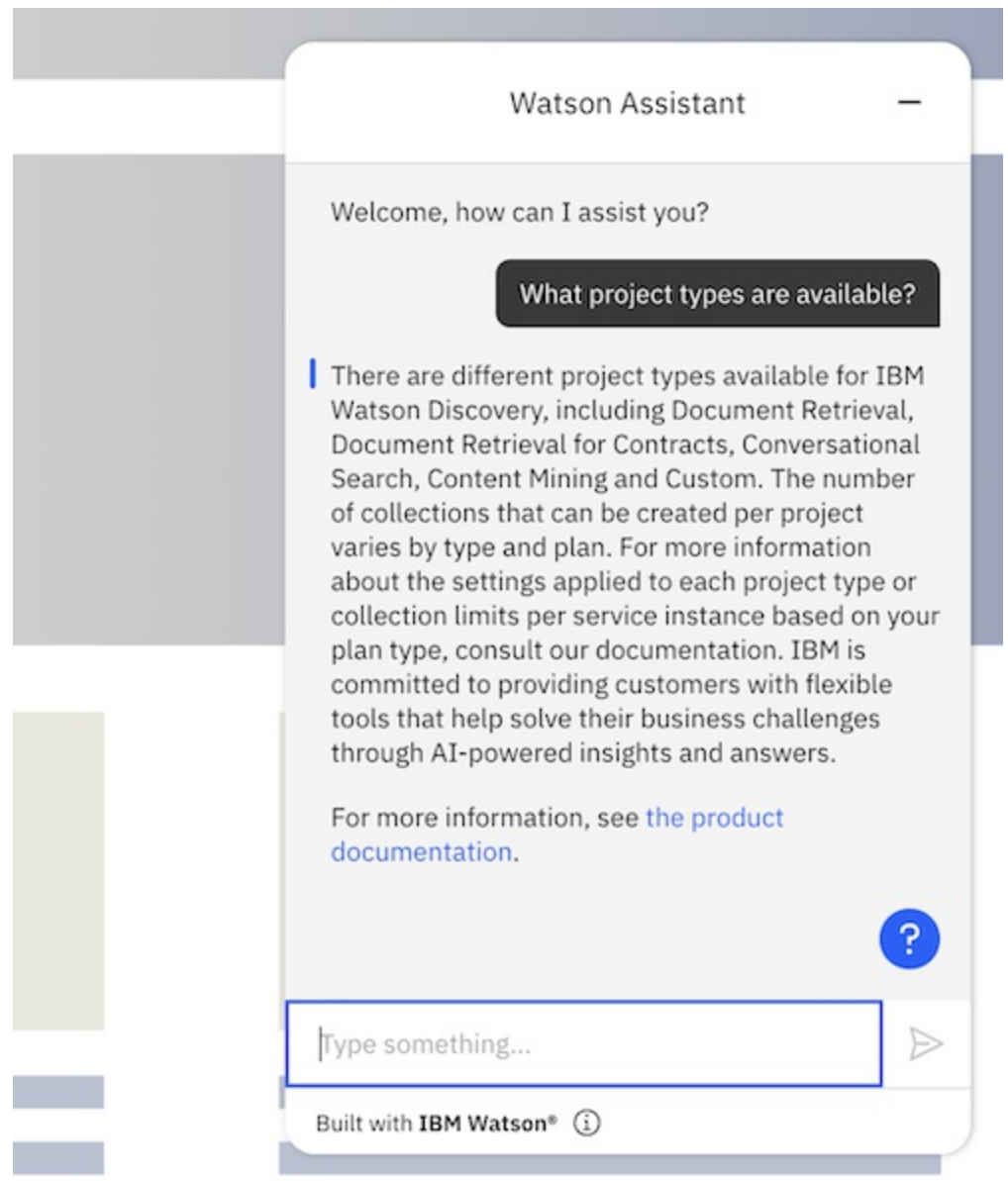


Web chat welcome message

3. Enter the following text question:

```
What project types are available?
```

The correct answer is returned and it includes a link to the product documentation.

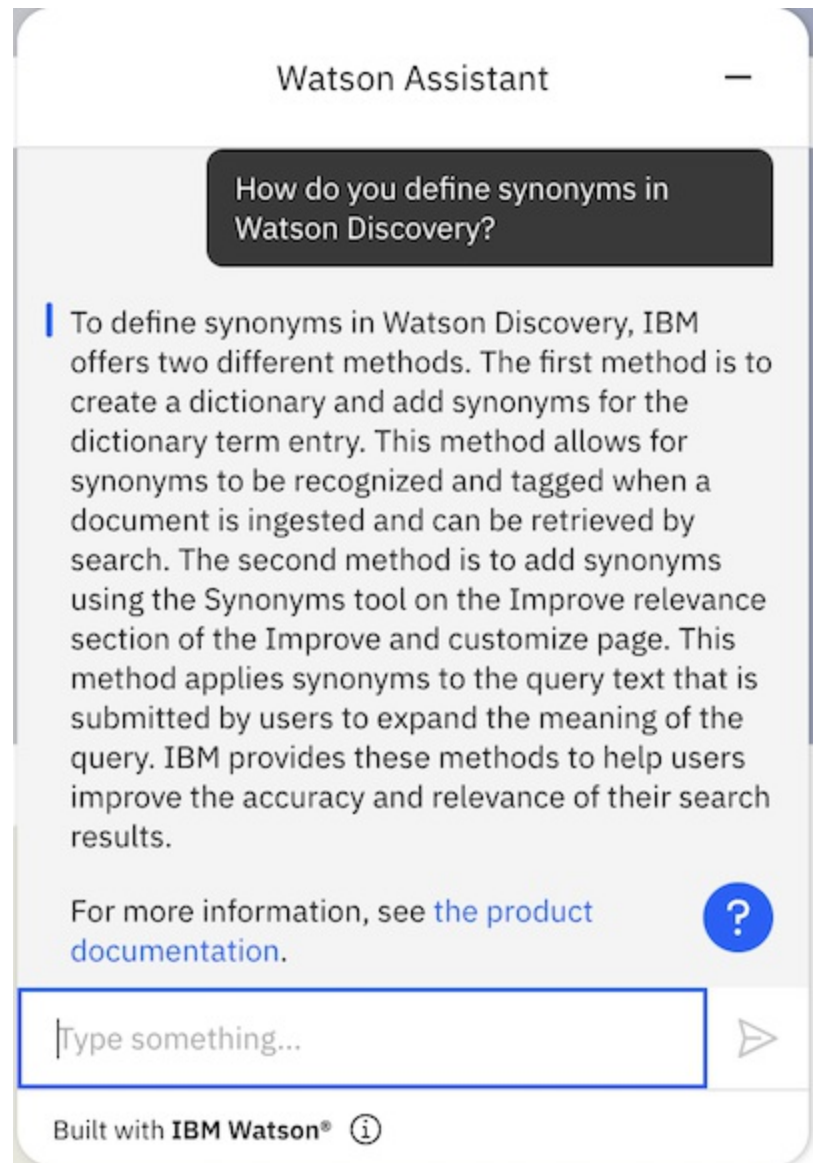


Web chat returns search response

4. Submit a question that wasn't used as a query example when you created the action.

   ```
   How do you define synonyms in Watson Discovery?
   ```

   A detailed answer is returned.



Web chat returns a detailed answer

5. Optionally ask the assistant other questions.

   If the assistant doesn't know the answer, reword the question to include “in Watson Discovery” to make it clearer that you are asking about how something works in Discovery specifically.

Congratulations! You successfully created an assistant that can answer questions about Discovery by retrieving information from the product documentation by way of the NeuralSeek service.

Summary

In this tutorial, you created a Watson Discovery Document Retrieval project with uploaded PDF files that contain the Discovery product documentation. Separately, you created a watsonx Assistant virtual assistant with a single action that can recognize user questions about Discovery. You added a custom extension to your assistant that connects to a third-party service called NeuralSeek that gets the correct answer from Discovery and rewords the response. Finally, you tested your virtual assistant by asking a question and getting an accurate and well-written response.

Next steps

The assistant that you created is available from the draft environment. Next, you can publish your assistant to a production environment and deploy it. You can deploy the assistant in various ways. For more information, see Overview: Previewing and publishing.

Building your assistant with actions

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Building actions from templates

When you create actions, you can choose a template that relates to the problem you're trying to solve. Templates help tailor your actions to include items specific to your business need. The examples in each template can also help you to learn how actions work.

Templates are available for different use cases, for example, booking a meeting, creating a support ticket, or making a payment. Templates include all the pieces that make up an action, such as steps, conditions, and different response types to collect customer answers.

Note: You can use templates in English-language assistants only.

These features are included as examples in action templates:

Feature	Description	More information
Step conditions	A step condition is a Boolean test, based on a runtime value. The step is used only if the test evaluates as true.	Adding conditions to a step
Options synonyms	Synonyms are variations of an option value that customers might enter.	Options

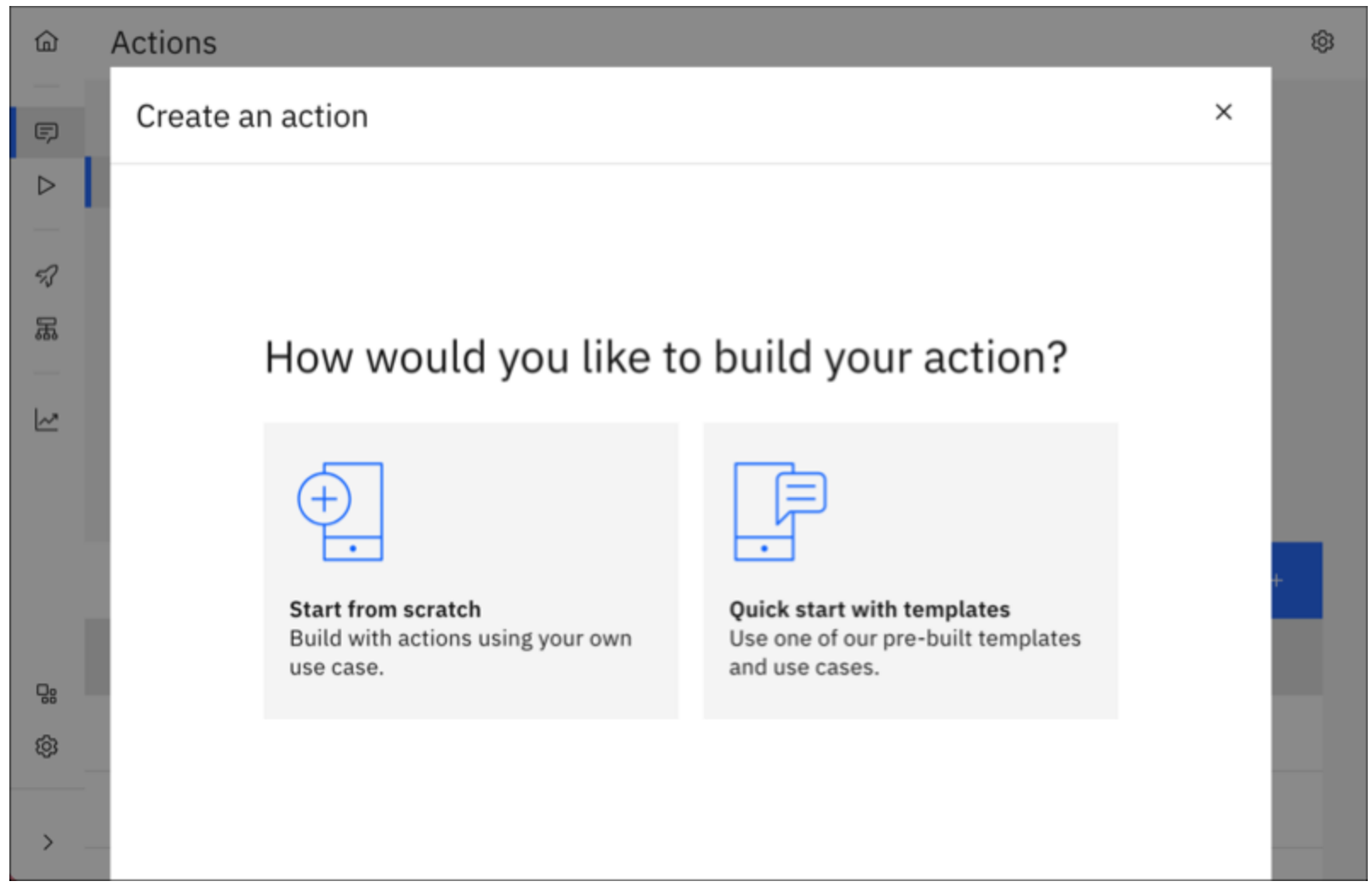
Customized response validation	When you edit a step that expects a customer response, you can customize how validation errors are handled.	When your customer gives invalid answers
Action variables	When customers reply to your assistant, they share information about themselves and what they want. Your assistant remembers this information, and other information about a conversation, as variables.	Using variables to manage conversation information
Session variables	A value that is not necessarily tied to a particular action can be stored as a session variable. Session variables are long-term memory: they persist throughout the user's interaction with the assistant, and your assistant can reference them from any action.	Using variables to manage conversation information
Regex response type	A regex response collects a text string that matches a pattern that is expressed as a regular expression. Use this response to capture a value that must conform to a particular pattern or format, such as an email address or telephone number.	Regex
Yes/No response type	A confirmation response presents customers with the choices of either Yes or No as buttons.	Confirmation
Connect to agent	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers the conversation to a live agent when necessary.	Connecting to a live agent
Embedded video	Includes a video to display a how-to demonstration, promotional clip, or other video content. In the web chat, a video response renders as an embedded video player.	Adding a Video response
Customer responses referenced in URLs	In your assistant's output, you can reference variables to personalize a URL link, including information specific to the customer such as account number or email address	Adding and referencing variables
Customized agent hand off information	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers the conversation to a live agent when necessary. You can customize messages the assistant displays as part of the transfer	Connecting to a live agent

Features

Creating actions from templates

To create actions from templates:

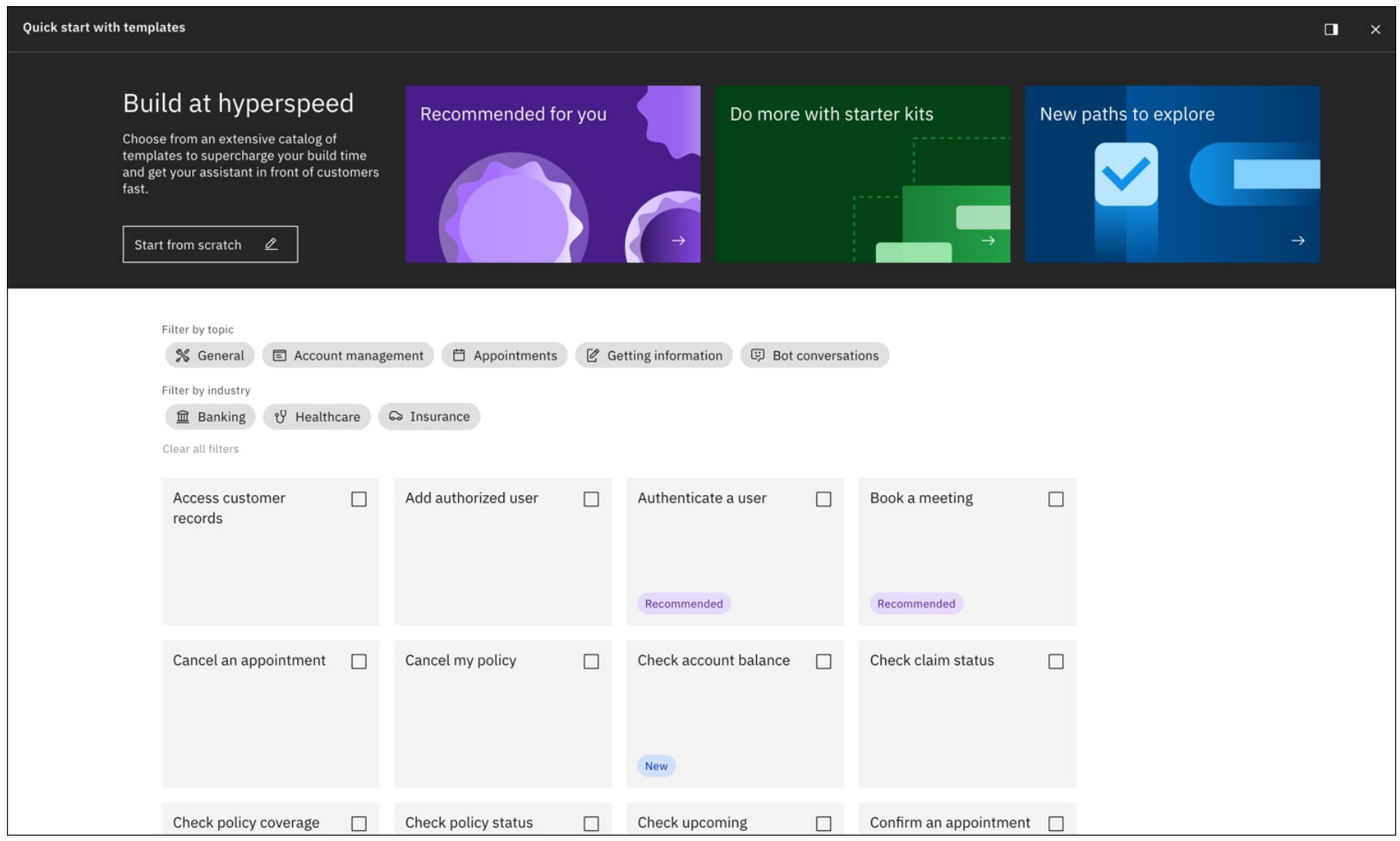
1. Open the **Actions** page.
2. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
3. On **Create an action**, choose **Quick start with templates**.



Create an action

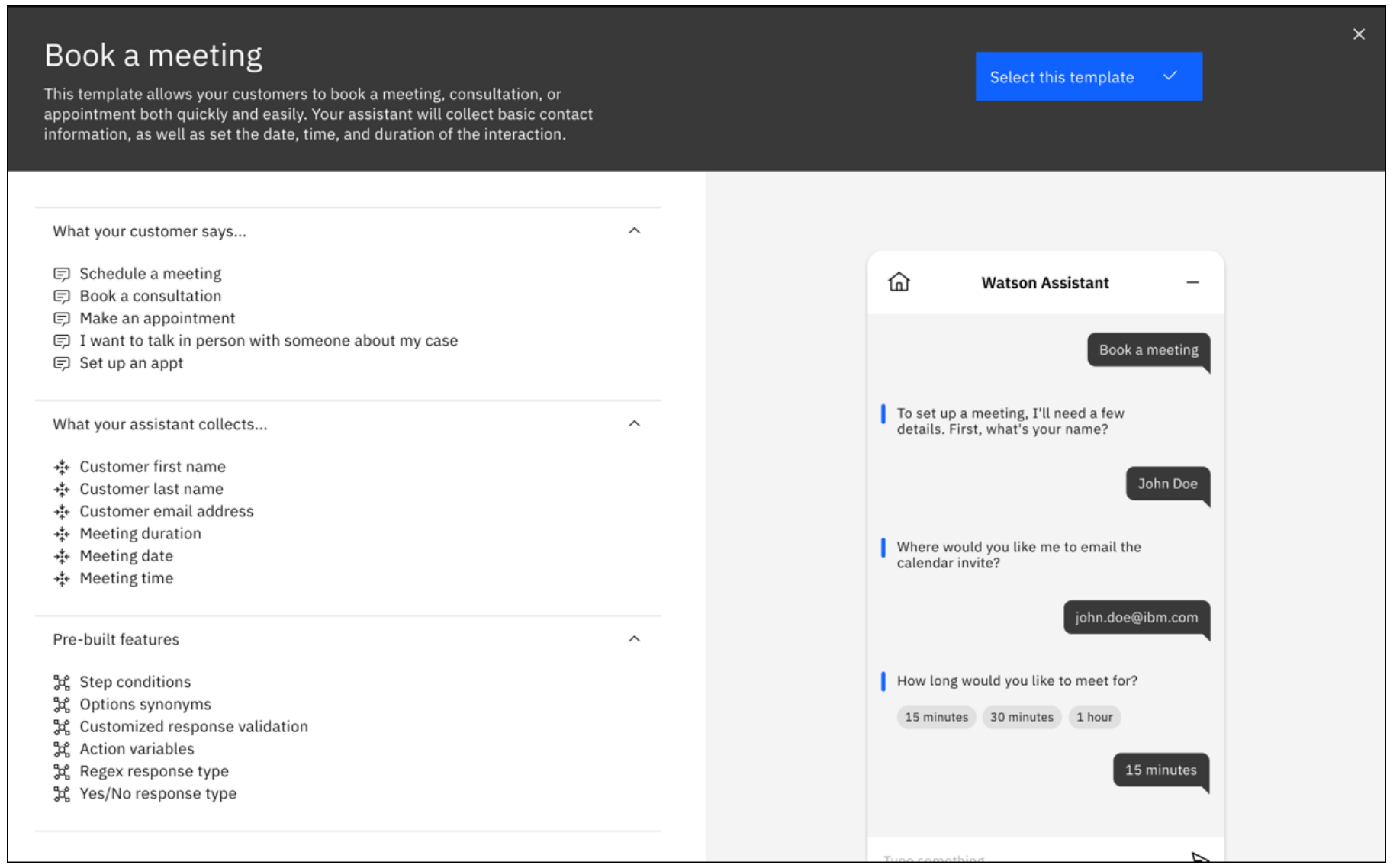
4. On **Quick start with templates**, click a template to read details about what it does. Or, you can click the checkbox to select the template right away.

> ⚠ **Important:** If you choose a starter kit template, these actions require you to add a custom extension before you add the actions from the starter kit. For more information, see Starter kit extension setup.



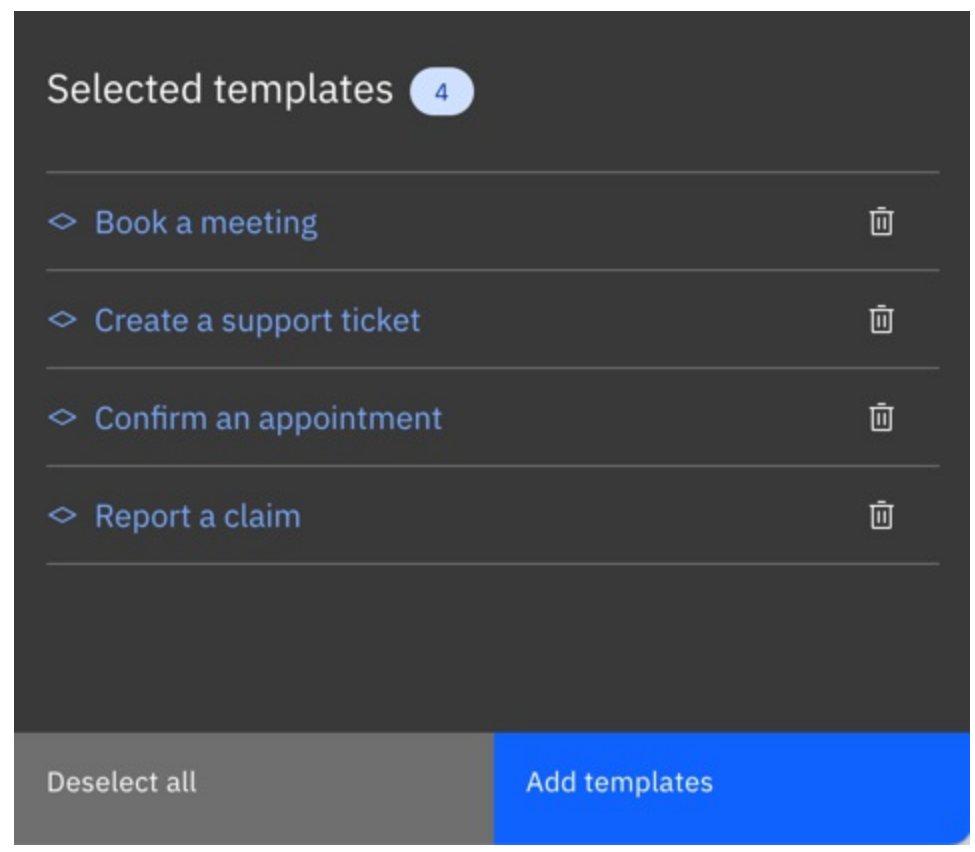
Quick start with templates

5. Review the details of the template, then click **Select this template**.



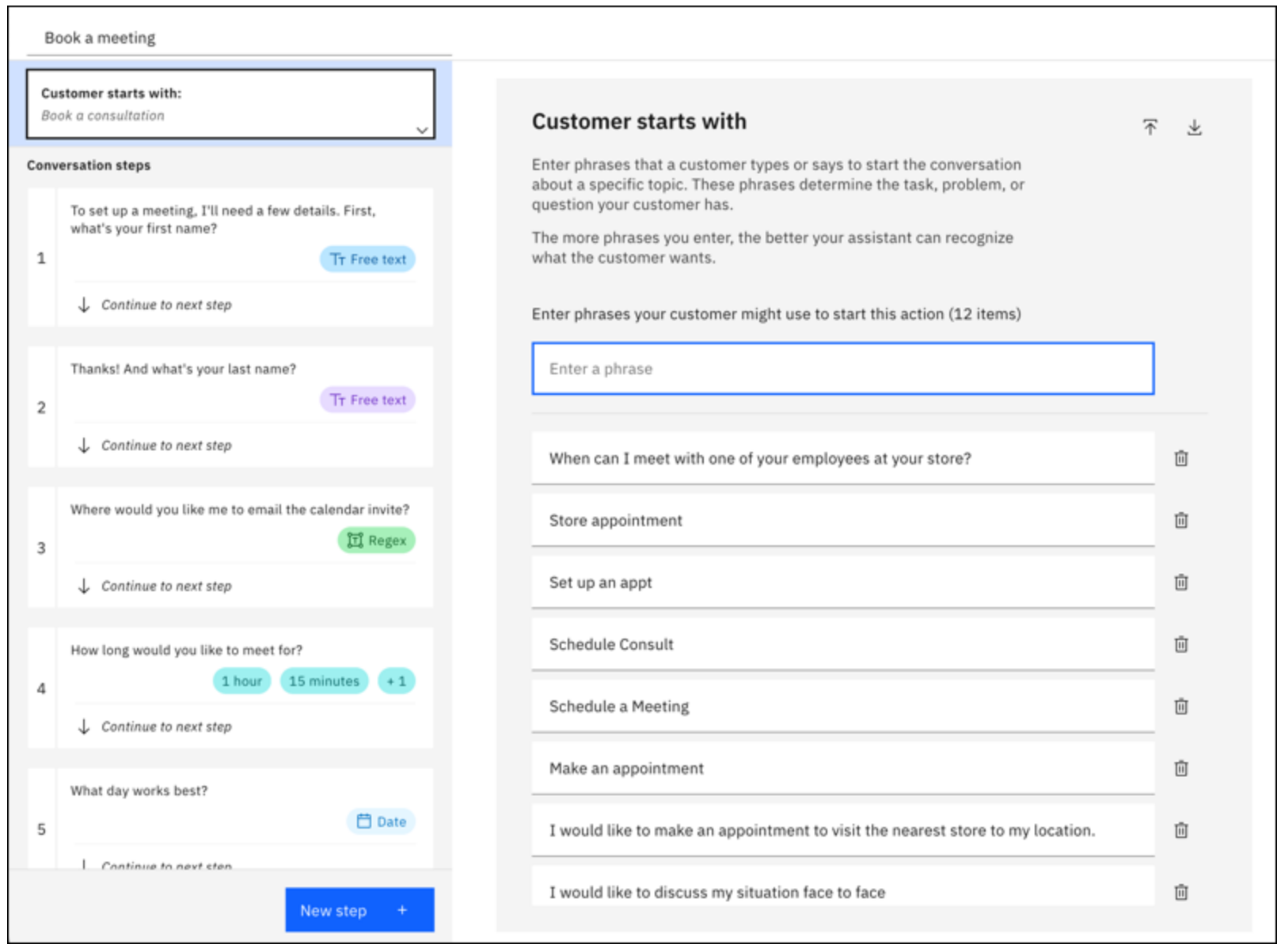
Template details

6. Your list of selected templates includes all the ones that you chose. You can select as many templates as you want. If you change your mind, click the trash can icon to remove a template from the list.



Selected templates

7. When you're ready, click **Add templates**.
8. In the actions editor, you can use a new action as-is, or modify it to fit your use case.



New action

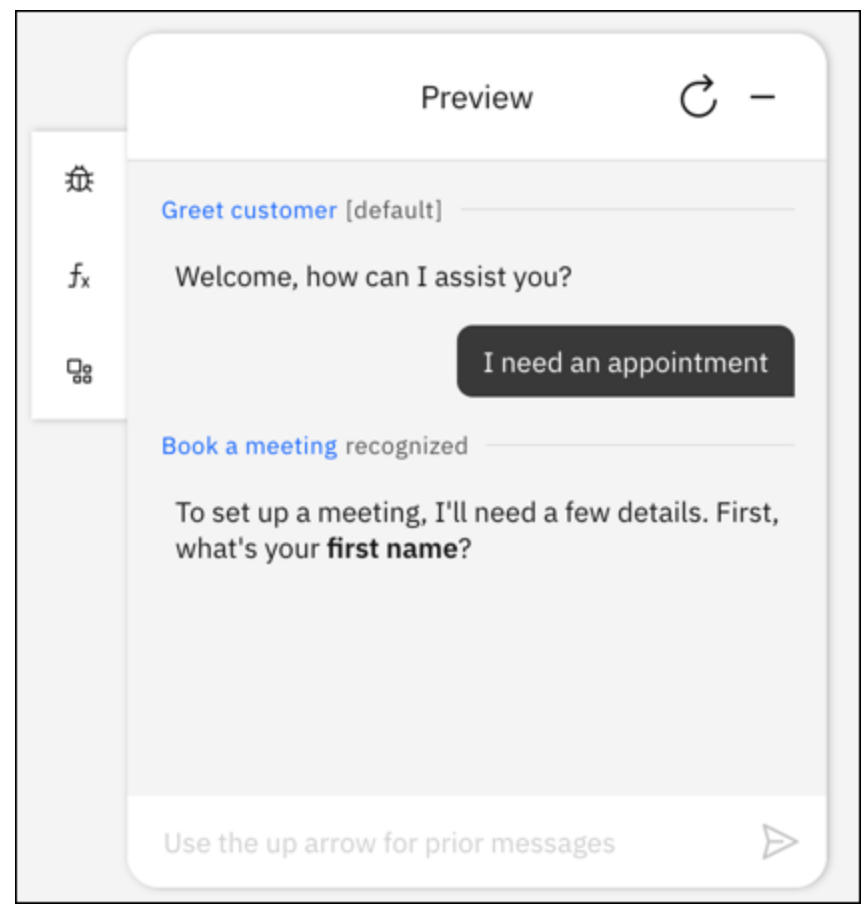
Note: You can create multiple actions from the same template. For example, if you used the **Book a meeting** template to create an action, you can choose that template again. If the first action is still named **Book a meeting**, the new action is added with the name **Book a meeting (1)**.

Preview actions

Most action templates are complete. (Starter kit templates are marked with an asterisk and require setup.) You can try out an action right away even without making any changes.

To preview the action that you created from a template:

1. In the actions editor, click **Preview**.
2. Try some example phrases from **Customer starts with** to see how the assistant responds.



Preview

Starter kit extension setup

Starter kit templates add actions that can use extensions to connect to data and systems outside of watsonx Assistant. To learn more about extensions in general, see Build a custom extension.

> ⚠ **Important:** It's important to add the extension to your assistant first, so the starter kit actions can detect the installed extension. Don't add the starter kit actions without adding the extension first.

This table lists each starter kit, a link to download an OpenAPI specification file that you need to set up the extension, and a link to setup instructions.

Starter kit	OpenAPI specification file	Setup instructions
Coveo search	coveo.openapi.json	Coveo search setup
Google custom search	google-custom-search-openapi.json	Google custom search setup
HubSpot	hubspot.advanced.openapi.json	HubSpot setup
NeuralSeek	Requires a file specific to your instance of NeuralSeek. Refer to the setup instructions.	NeuralSeek extension setup
ServiceNow	sn.openapi.json	ServiceNow extension setup
Zendesk	zendesk-openapi.json	Zendesk extension setup

Starter kits

Coveo search extension setup

To set up the extension for Coveo search, see Coveo search extension setup.

Google custom search extension setup

To set up the extension for Google custom search, see Google custom search extension setup.

HubSpot extension setup

To set up the extension for HubSpot:

1. In HubSpot, you need to create a developer account and then create a private app. The private app includes an access token that you need for authentication. For detailed instructions, see Getting private app access token in the HubSpot Custom Extension starter kit GitHub repository.
2. Your HubSpot account needs custom properties. Follow the instructions in Adding Custom Properties in HubSpot in the HubSpot Custom Extension starter kit GitHub repository.
3. Download the OpenAPI specification file: hubspot.advanced.openapi.json.
4. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
5. After you build the HubSpot custom search extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your HubSpot private app access token to authenticate. For general instructions on adding any custom extension, see Adding an extension to your assistant.
6. Open the **Actions** page.
7. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
8. On **Create an action**, choose **Quick start with templates**.
9. On **Quick start with templates**, add the HubSpot starter kit.

NeuralSeek extension setup

To set up the extension for NeuralSeek, see NeuralSeek extension setup.

ServiceNow extension setup

To set up the extension for ServiceNow:

1. Request a ServiceNow Personal Developer Instance. For detailed instructions, see Personal Developer Instances.
2. Download the OpenAPI specification file: sn.openapi.json.
3. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
4. After you build the ServiceNow extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your ServiceNow admin username and password to authenticate. Replace the default instance server variable with your own. For general instructions on adding any custom extension, see Adding an extension to your assistant.
5. Open the **Actions** page.
6. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
7. On **Create an action**, choose **Quick start with templates**.
8. On **Quick start with templates**, add the ServiceNow starter kit.

Zendesk extension setup

To set up the extension for Zendesk:

1. In Zendesk, open the **Admin Center**, for example, `https://{server-domain}.zendesk.com/admin`.
2. In **Apps and integrations**, click **Zendesk API**.
3. On the **Settings** tab, enable **Password access**.
4. Download the OpenAPI specification file: zendesk-openapi.json.
5. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
6. After you build the Zendesk extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Zendesk username and password to authenticate. For general instructions on adding any custom extension, see Adding an extension to your assistant.
7. Open the **Actions** page.
8. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
9. On **Create an action**, choose **Quick start with templates**.
10. On **Quick start with templates**, add the Zendesk starter kit.

Starting the conversation

As you start building your assistant, one of the first things you should consider is how it will start each new conversation with a user. This might be as simple as saying hello, or it might involve asking some questions to gather data the assistant needs before it can do anything else. Starting a new conversation is handled by the *Greet customer* action, which is automatically created (with a default greeting) when you create an assistant.

> **Note:** The *Greet customer* action is triggered in situations where your assistant initiates the conversation and then waits for input from the user. Depending on how users connect to your assistant, the Greeting action might not be triggered. (For more information, see When the greeting action is triggered.)

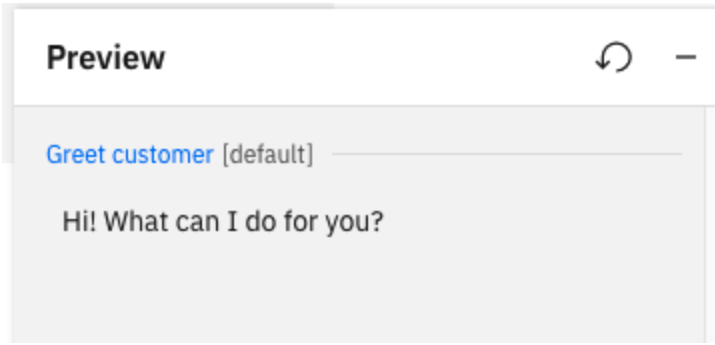
Customizing the greeting

The *Greet customer* action is automatically provided for any assistant. To customize this action:

1. In the left-hand navigation pane of the actions editor, click **Actions** to expand the list.
2. Click **Set by assistant** to see a list of built-in actions that are automatically provided when you create an assistant.
3. Click *Greet customer* to edit the greeting action.

   Notice that for this action, you cannot specify the customer input that starts the action. That's because the greeting action is automatically sent when the assistant starts the conversation, before any user input is received.
4. Under **Conversation steps**, click the first step. In the **Assistant says** field, edit the default text to specify the greeting text you want to use. For example, if you want your assistant to use a more casual tone, you might specify the text `Hi! What can I do for you?`

5. To see your new greeting in action, click **Preview**. In the Preview pane, you should see your customized greeting appear.



Adding a mandatory welcome flow

Apart from how it is initiated, the *Greet customer* action is just like any other action. If you need to start each conversation with more than just a standard greeting, you can configure your *Greet customer* action with more steps and customer responses, just as you would with any other action. For example, instead of just saying hello, your *Greet customer* action might start by asking for the user's account number. For more information about editing actions, see Overview: Editing actions.

You can also use the *Greet customer* action to initialize variables for use throughout the conversation. For example, you might want to initialize tracking variables at the beginning of the conversation, or store the user's name so you can personalize the assistant's interactions. In general, you can set variables in the *Greet customer* action just as you can with any other action. (For more information about setting variables, see Managing information during the conversation.)

Keep in mind that you should not rely on the *Greet customer* action to initialize required variables unless you are certain it will always be triggered at the beginning of each conversation. For more information, see When the greeting action is triggered.

When the greeting action is triggered

Depending on how you publish your assistant, the *Greet customer* action might not be triggered. This action is triggered only when the assistant, rather than the user, starts the conversation, as in the following situations:

- Assistant preview
- Phone integration
- Web chat integration with home screen disabled
- A custom client application, depending on design

In this kind of situation, the integration or client application starts the session by sending an empty message to the assistant and waits for the assistant to greet the user (this is when the *Greet customer* action is triggered).

However, there are other situations in which the *Greet customer* action is never triggered:

- Integrations with text messaging channels, such as Slack, Facebook Messenger, or SMS. With these kinds of channels, the user starts the conversation by sending an initial message or request. This triggers the appropriate action for handling the user's request, so the *Greet customer* action is not triggered.
- Web chat integration with home screen enabled. The home screen is an optional feature of the web chat integration. When enabled, the home screen displays a welcome message to the user; because the greeting is defined in the web chat configuration, the *Greet customer* action is not triggered.

Understanding your users' questions or requests

Actions represent the tasks or questions that your assistant can help customers with. Each action has a beginning and an end, making up a conversation between the assistant and a customer. Learn how to begin an action, where it understands and recognizes a goal based on the words a customer uses to ask a question or make a request.

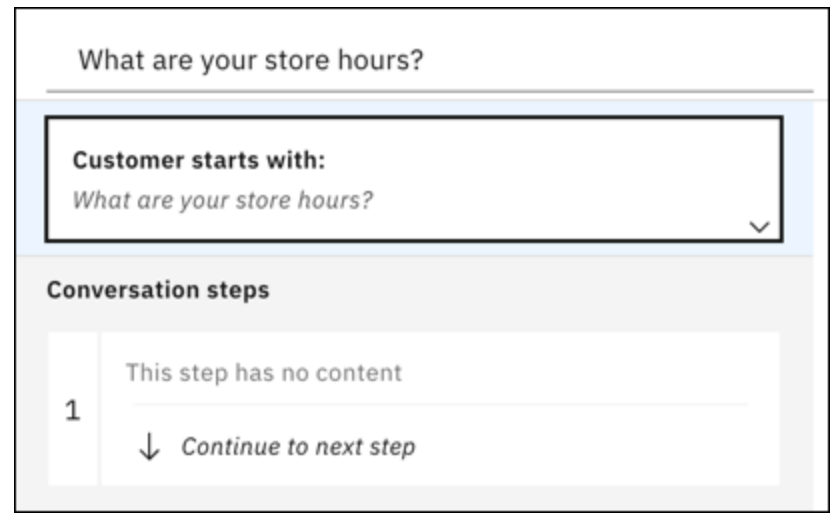
Beginning an action

Each assistant can include as many actions as you need to have conversations with your users. You design each individual action to recognize a specific question or request, and when it does, the action starts.

When you create a new action, your first task is to enter one phrase that a customer types or says to start the conversation about a specific topic. This phrase determines the problem that your customer has or the question your user asks.

To get going, you need to enter only one phrase, for example: `What are your store hours?`.

After you enter the phrase, it is stored in **Customer starts with**, at the start of the action.

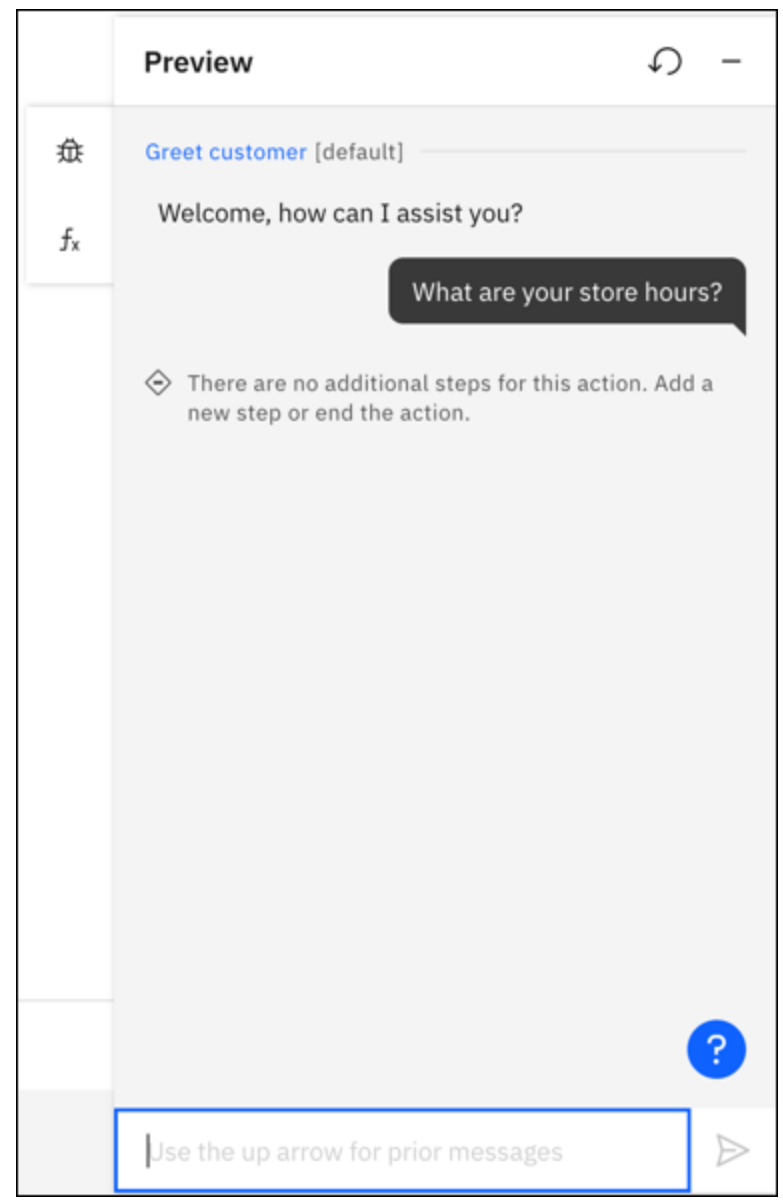


Customer starts with

Testing your phrase

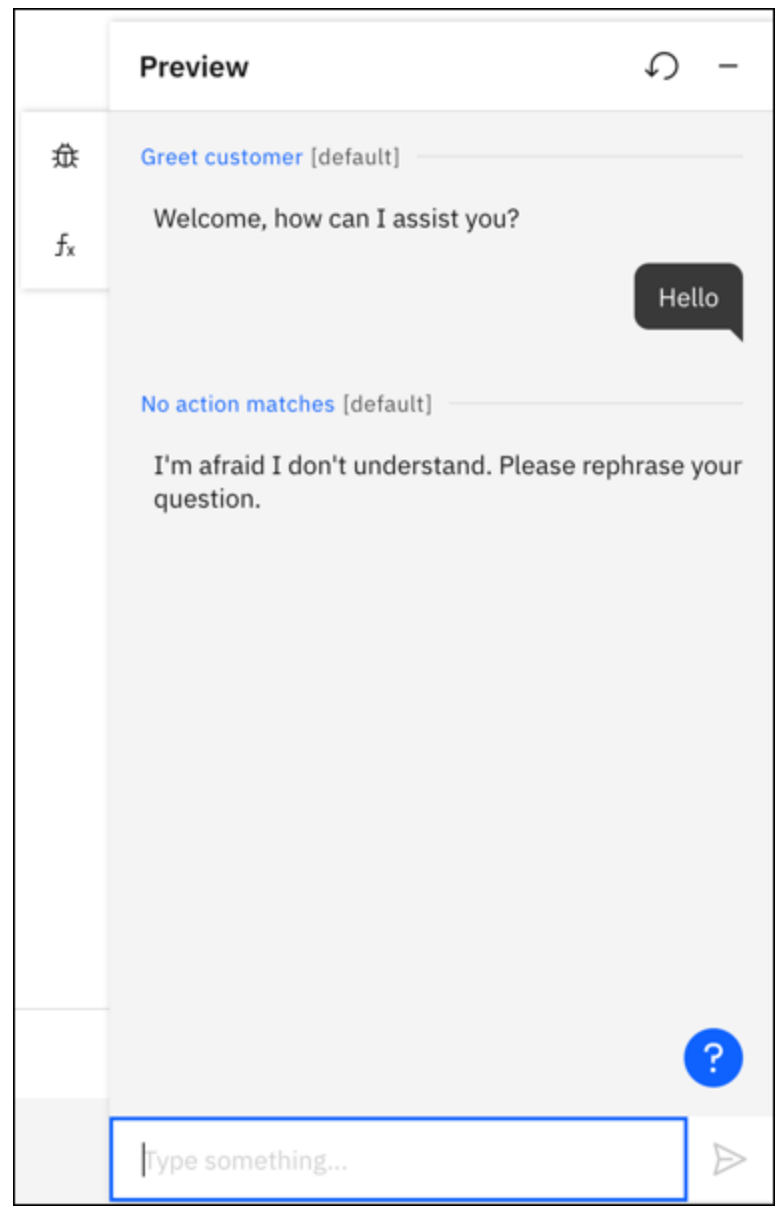
Before even doing anything else with your action, you can already start checking that your assistant recognizes the starting phrase.

1. Click **Preview**.
2. Enter your first phrase, for example: `What are your store hours?`.
3. If you see `There are no additional steps for this action` that means the action recognizes the phrase. (And it's because you didn't add anything else to your action.)



Preview

4. If the assistant doesn't understand the phrase, you see the built-in action `No matches`. For more information, see [When the assistant can't understand your customer's request](#).



No matches

Adding more examples

When you're creating a new action, one example phrase is enough to start with. You can build the rest of your action with steps before you add more example phrases. Then, return to **Customer starts with** and add 10 or more variations of the same question or request, by using words that your customers commonly use. For example:

- `Are you open on the weekend?`
- `How late are you open today?`
- `Get store hours`
- `What time do you open?`
- `Are you open now?`

Each phrase can be up to 1,024 characters in length.

By adding these phrases, your assistant learns what is the right action for what a customer wants. The additional examples build the training data that the machine learning engine of watsonx Assistant uses to create a natural language processing model. The model is customized to understand your uniquely defined actions.

Uploading phrases

If you have many example phrases, you can upload them from a comma-separated value (CSV) file than to define them one by one. If you are migrating intent information from the classic experience to example phrases in watsonx Assistant, see Migrating intents and entities.

1. Collect the phrases into a CSV file. Save the CSV file with UTF-8 encoding and no byte order mark (BOM).
    - If you are creating a new CSV file to upload phrases, the format for each line in the file is as follows:

      ```
      $ <phrase>
      ```

      where `<phrase>` is the text of a user example phrase. If you're using a spreadsheet to create a CSV file, put all your phrases into column 1, as shown in the following example:

	A	B
1	Tell me the current weather conditions.	
2	Is it raining?	
3	What's the temperature?	

Example spreadsheet

- If you downloaded intents from the classic experience, the format for each line in the file is as follows:

```
$ <phrase>,<intent>
```

Where `<phrase>` is the text of a user example phrase, and `<intent>` is the name of the intent. For example:

```
$ Tell me the current weather conditions.,weather_conditions
Is it raining?,weather_conditions
What's the temperature?,weather_conditions
```

> ⚠ **Important:** Only one intent can be uploaded per action, so the `<intent>` information listed in the second column of the CSV file must be the same.

2. Go to **Customer starts with** at the start of the action.
3. Click the **Upload** icon .
4. Select a file from your computer.

   The file is validated and uploaded, and the system trains itself on the new data.

Downloading phrases

You can download your example phrases to a CSV file, so you can then upload and reuse them in another watsonx Assistant application.

1. Go to **Customer starts with** at the start of the action.
2. Click the **Download** icon .

   Your example phrases are downloaded to a CSV file.

Asking clarifying questions

When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. Instead of guessing which action to take, your assistant shows a list of the possible actions to the customer, and asks the customer to pick the right one.

Sample conversation

Any **Created by you** action that might match the customer's input can be included in the choices that are listed by a clarifying question. The **Set by assistant** actions are never included.

In the assistant output, the possible actions are listed by name. The default name for an action is the text of the first example message that you add to it (such as `I want to open an account`), but you can change this name to something more descriptive.

The order in which the actions are listed might change. In fact, the actions themselves that are included in the list might change. This behavior is intended.

As part of development that is in progress to help the assistant learn automatically from user choices, the actions that are included and their order in the list is randomized on purpose. Randomizing the order helps to prevent bias that can be introduced by a percentage of people who always pick the first option without carefully reviewing all of their choices beforehand.

Customizing clarifying questions

To customize clarification, you can:

- Change settings like the wording your assistant uses to introduce the clarification list or when no action matches.
- Enable *response modes* to modify the assistant's behavior when it asks questions. For more information, see Response modes.

To change settings, complete the following steps:

1. From the **Actions** page of the assistant, click **Global settings**.
2. On the **Clarifying questions** tab, you can edit the **Ask clarifying questions** section:

Field	Default text	Description
Assistant says	`Did you mean:`	The text that is displayed before the list of clarification choices. You can change it to something else, such as `What do you want to do?` or `Pick what to do next`.
No matches	None of the above	The choice for customers when none of the other choices are right. You can change it to something else, such as `I need something else` or `These aren't what I want`. Or, you can remove the text to omit offering this choice. The result of this choice is different depending on the state of the action. If an action has not been triggered yet, the assistant uses the No matches action. If an action has been triggered and has not ended yet, the assistant uses the validation settings for the step.

Ask clarifying question settings

3. If you enable response modes, you can modify this text:

Field	Default text	Description
One action matches	`Something else`	If an assistant prioritizes one action that it thinks matches the customer need, it can clarify the match by asking the customer to confirm. This choice accompanies the single action in case the customer needs something else. You can change it to something else, such as `I need something else` or `This isn't what I want`.
Connection to support	Connect to support	The assistant can include a choice to connect to other support in the list of clarifying questions. If the customer picks this choice, the assistant uses your *Fallback* action. You can change it to something else, such as `Talk to a live agent` or `Search for the answer`.

Response modes settings

4. Click **Save**, and then click **Close**.
5. Publish a new version of your assistant to the live environment to apply the customizations. For more information, see Publishing your content.

Disabling clarifying questions

You can disable clarifying questions for all actions.

To disable clarification for all actions:

1. From the **Actions** page of the assistant, click **Global settings**.
2. On the **Clarifying questions** tab, ensure that the **Response modes** switch is set to **Off**.
3. Set the **Enable disambiguation** switch to **Off**.
4. Click **Save**, and then click **Close**.
5. Publish a new version of your assistant to the live environment to disable clarification. For more information, see Publishing your content.

Excluding an action from clarifying questions

You can also prevent a single action from being used in a clarifying question. The effect of this choice depends on the confidence score for the action that you exclude.

If the action has the highest confidence score for a customer's question, no clarifying question is asked, and the action is triggered.

If the action doesn't have the highest confidence score, the action is excluded from the list of choices in the clarifying question.

For more information about confidence scores, see Confidence scoring.

To exclude an action from clarification:

1. From the action editor, click the **Action settings** icon.
2. In Action Settings, toggle the **Ask clarifying question** switch to **Off**.

Coordinating how multiple actions start

As you work on your assistant, it's a good idea to coordinate customer phrase examples across multiple actions. It's important to distinguish how each action is triggered. When a user enters a question or request, the phrase is evaluated across all the **Customer starts with** examples in every action. If two actions have similar phrase examples, then the wrong action might get triggered by your user's question.

Confidence scoring

Behind the scenes, watsonx Assistant determines a confidence score for each phrase. The score is absolute, meaning that a confidence score is assigned based on a predetermined scale, and not relative to other customer phrases. This approach adds flexibility in case multiple questions or requests are detected in a single user input. It also means that the system might not trigger an action at all, if a phrase has a low confidence score. As confidence scores change, your action examples might need restructuring.

To learn more about review and testing confidence scores, see Action confidence score in Reviewing and debugging your work.

Boost words

Beta

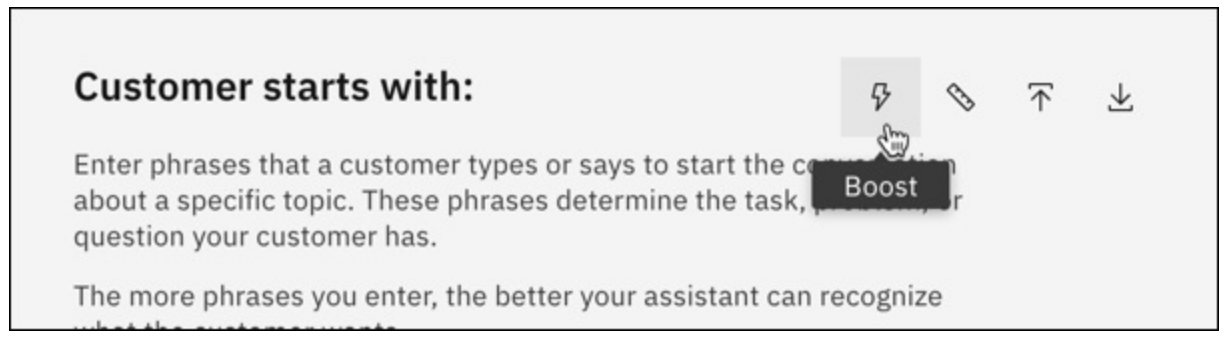
Use *boost words* to enter keywords or phrases to help the assistant recognize what a customer wants. If a customer's input includes matches the boost words or phrases, the assistant can be more confident to use an action.

Boost words or phrases are used in addition to example phrases. They can contain only letters, underscores, hyphens, spaces, or dots. They must start with an alphanumeric character. Words or phrases can't be entered more than once per action, but you can use the same boost words or phrases across multiple actions.

This beta feature is available for evaluation and testing purposes only. To request access to the *Boost words* feature, use the Request access for watsonx Assistant Boost Words (beta) form.

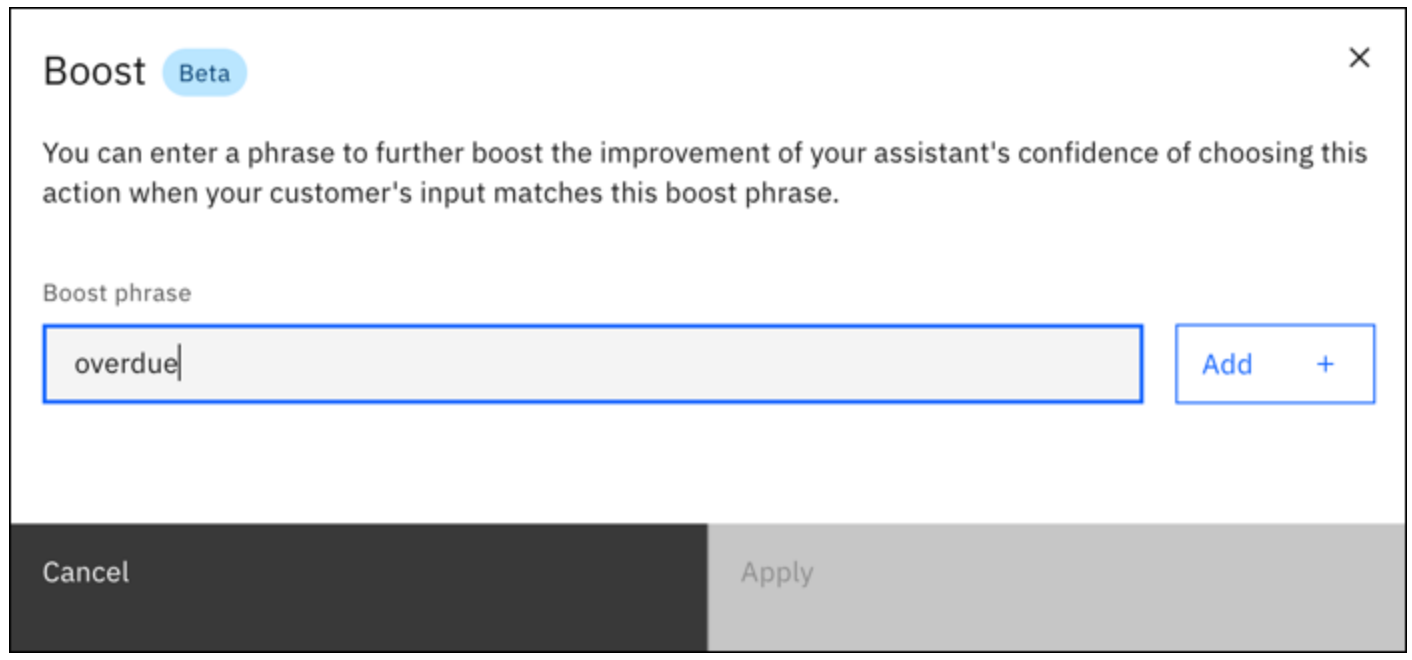
To add boost words or phrases to an action:

1. Create or open an action.
2. In **Customer starts with**, ensure that the action has at least one example phrase before you add any boost words.
3. Click the **Boost** icon.



Boost icon

4. Enter a boost word or phrase, then click **Add**.

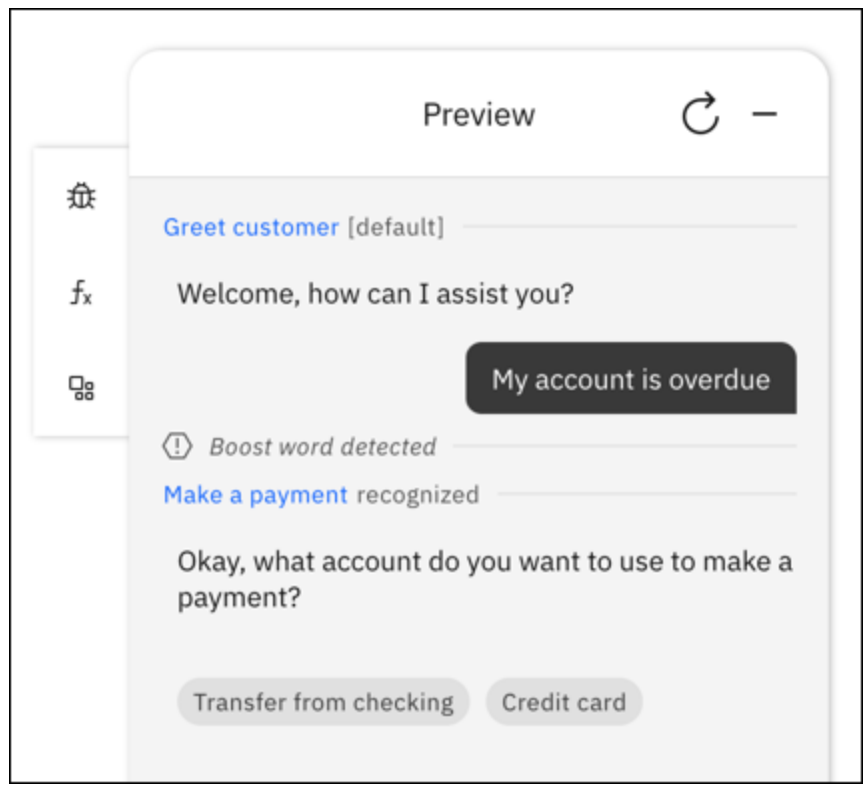


Boost words

5. You can keep entering words or phrases one at a time. When you're finished, click **Apply**.

Testing recognition of boost words

You can use **Preview** to test recognition of boost words. If a boost word or phrase matches, you can see `Boost word detected` in the Preview pane.



Test boost in preview

Adding assistant responses

When an action is activated, the body of the action is composed of multiple *steps* that make up the conversation between your assistant and your users. One part of each step is what the assistant says to the customer when the step is processed.

To create your assistant's response in a step, you use the **Assistant says** section. This represents the text or speech that the assistant delivers to a user at a particular step. Depending on the step, you can add a complete answer to a user's question or ask a follow-up question.

You can enter a simple text response just by entering the text that you want your assistant to display to the user. You can also add formatting and web content, and you can reference user information by using *variables*.

Formatting responses

Use the text editor tools to apply font styling, such as bold or italic, to the text or to add links.

Behind the scenes, font styling and link syntax are stored in Markdown format. If you are using the web chat integration, HTML and Markdown tagging are supported. For more information, see Markdown formatting.

HTML tags (except for links) are automatically removed from text responses that are sent to the Facebook, WhatsApp, and Slack integrations, because those channels do not support HTML formatting. HTML tags are still handled in channels that support them (such as the web chat) and stored in the session history.

Note: If you're using a custom client application that does not support Markdown, don't apply text styling to your text responses.

Adding and referencing variables

During the conversation, your assistant stores information as *variables*. Variables are containers for data values that become available at run time. The value of a variable can change over time. Variables include *action variables*, which persist only during a particular action, and *session variables*, which are available to any action. For more information about variables, see Managing information during the conversation.

In your assistant's output, you can reference variables to personalize the conversation or include information that is available at run time. For more information about referencing variables in what your assistant says, see Using variables to customize the conversation.

Testing responses

To check that the assistant responses are formatted correctly, you can use **Preview**.

1. Click the **Preview** button.
2. To start the action, enter your first phrase, for example: `What are your store hours?`.
3. When the assistant responds, check that the message displays as you intended with formatting and use of variables.

Tips for adding responses

- Keep answers short and useful.
- Reflect the user's intent in the response. Doing so assures users that the bot is understanding them, or if it is not, gives users a chance to correct a misunderstanding immediately.
- Include links to external sites in responses if the answer depends on data that changes frequently.
- Word your responses carefully. You can change how someone reacts to your system based on how you phrase a response. Changing one line of text can prevent you from having to write multiple lines of code to implement a complex programmatic solution.

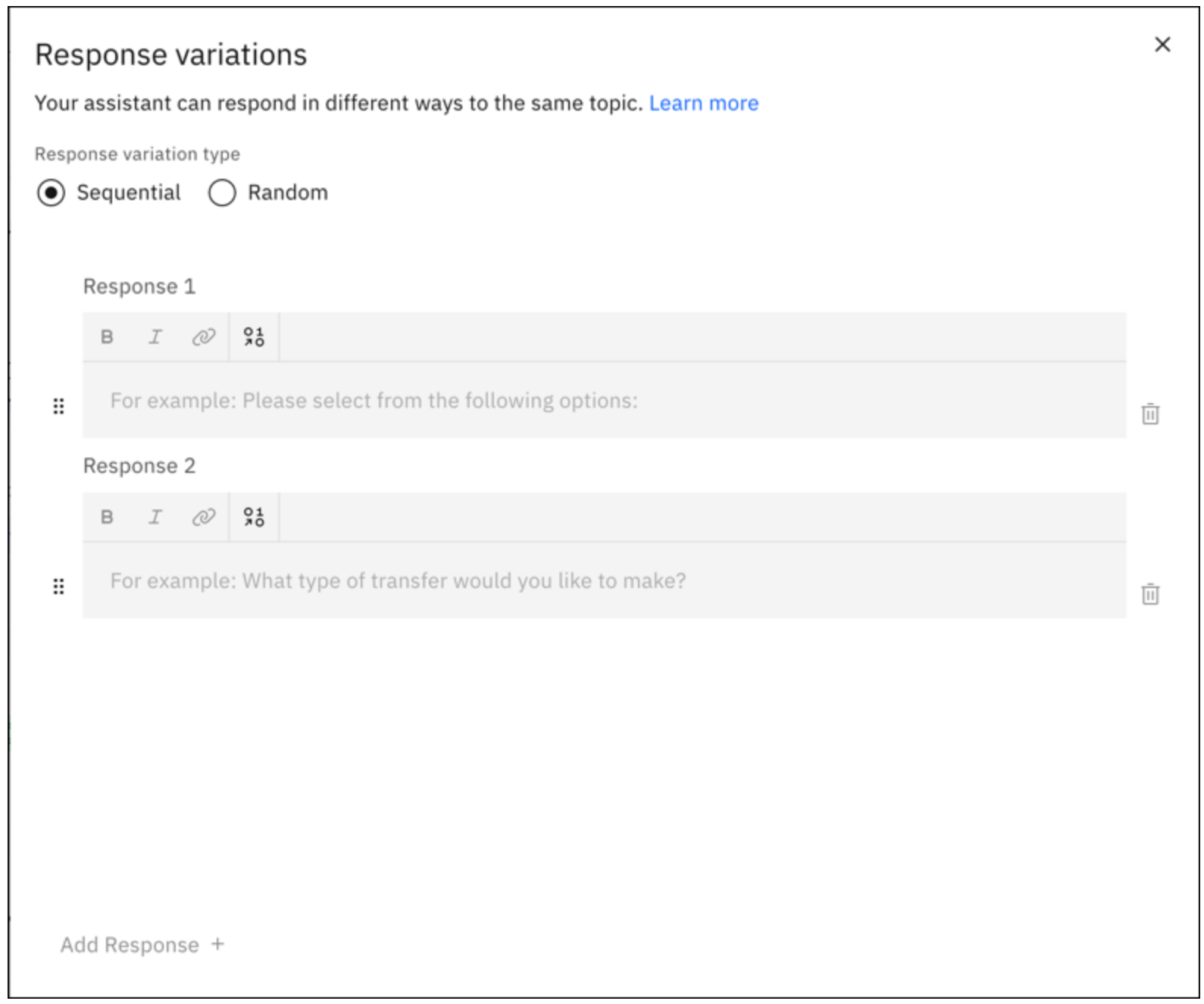
Adding variations

If your users return to your assistant frequently, they might be bored to see the same greetings and responses every time. You can add *response variations* so that your assistant can respond to the same request in different ways.

You can choose to rotate through the response variations sequentially or in random order. By default, responses are rotated sequentially, as if they were chosen from an ordered list.

To add response variations:

1. In **Assistant says**, click the **Add response variations** icon .
2. For **Response variation type**, choose whether to rotate through the response variations sequentially or in random order. For more information, see Sequential or random.

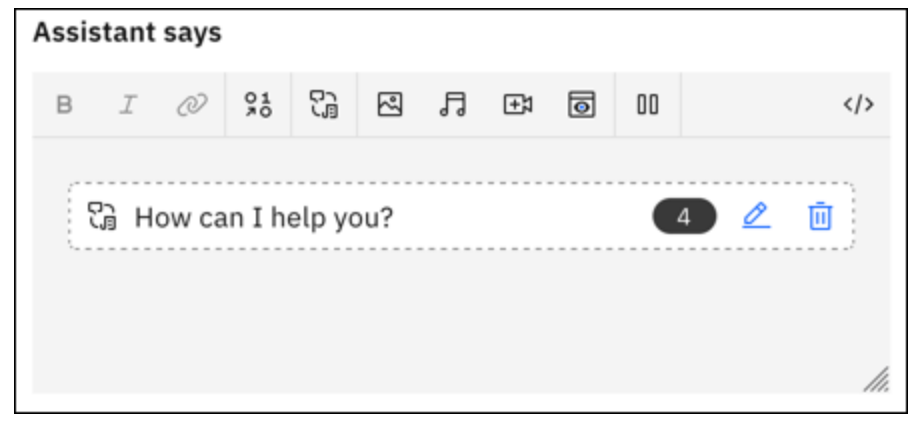


Response variations

3. Add each variation into its own field. For example:

Response number	Variation
Response 1	How can I help you?
Response 2	What can I do for you today?
Response 3	Tell me what I can help with.
Response 4	Can I help you?

1. When you're finished, click **Apply**. The variations appear as a block inside **Assistant says**. You can click the **Edit** icon to update the variations, or click the **Delete** icon to remove all the variations. Also, you can add multiple sets of response variations to a step.



Response variations in Assistant says

Sequential or random

For **Response variation type**, you can choose **Sequential** or **Random**.

Sequential returns the first response variation the first time the action is triggered, the second response variation the second time the action is triggered, and so on, in the same order as you entered the variations. This results in responses returned in the following order when the node is processed:

- First time:

```
$ How can I help you?
```

- Second time:

```
$ What can I do for you today?
```

- Third time:

```
$ Tell me what I can help with.
```

- Fourth time:

```
$ Can I help you?
```

Random selects variation the first time that the action is triggered, and randomly selects another variation the next time, but without repeating the same variation consecutively. This example show an order that responses might appear:

- First time:

```
$ Tell me what I can help with.
```

- Second time:

```
$ Can I help you?
```

- Third time:

```
$ How can I help you?
```

- Fourth time:

```
$ What can I do for you today?
```

Media responses

In addition to text responses, you can use other *response types* to send responses that include multimedia or interactive elements.

The action editor supports the following media response types:

- **Image**: Embeds an image into the response. The source image file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Video**: Embeds a video player into the response. The source video must be hosted somewhere, either as a playable video on a supported video streaming service or as a video file with a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Audio**: Embeds an audio clip into the response. The source audio file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **iframe**: Embeds content from an external website, such as a form or other interactive component, directly within the chat. The source content must be publicly accessible using HTTP, and must be embeddable as an HTML `iframe` element.

Different channel integrations have different capabilities for displaying media responses. To see which channel integrations support which response types, see Channel integration support for response types.

If you want to define different responses that are customized for different channels, you can do so by editing the response by using the JSON editor. For more information, see Targeting specific integrations.

By editing your responses in the JSON editor, you can also access more response types for handling channel-specific interactions.

Note: For more information about how to edit responses by using the JSON editor, see Defining responses using the JSON editor.

Adding an *Image* response

Add an *Image* response to display an image to the customer.

The *Image* response type is supported by the following channel integrations:

- Web chat
- SMS
- Slack
- Microsoft Teams
- Facebook
- WhatsApp

To add an *Image* response, complete the following steps:

1. In the **Assistant says** field, click the **Image** icon.
2. In the **Source URL** field, type the full URL to the hosted image.

   The image must be in `JPEG`, `GIF`, or `PNG` format. The image file must be stored in a location that is publicly addressable by an `https:` URL (such as `https://www.example.com/assets/common/logo.png`).

   To access an image that is stored in IBM Cloud® Object Storage, enable public access to the individual image storage object, and then reference it by specifying the image source with syntax like this: `https://s3.eu.cloud-object-storage.appdomain.cloud/your-bucket-name/image-name.png`.
3. Optionally specify an image title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the image.

   > **Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

4. Click **Apply**.

Adding an *Audio* response

Add an *Audio* response to include spoken-word or other audible content. In the web chat, an audio response renders as an embedded audio player. In the phone integration, an audio response plays over the phone.

The *Audio* response type is supported by the following channel integrations:

- Web chat
- Phone
- SMS
- Slack
- Facebook
- WhatsApp

To add an *Audio* response, complete the following steps:

1. In the **Assistant says** field, click the **Audio** icon.
2. In the **Source URL** field, type the full URL to the hosted audio clip:
   - To link directly to an audio file, specify the URL to a file in any standard format such as MP3 or WAV. In the web chat, the linked audio clip renders as an embedded audio player.
   - To link to an audio clip on a supported audio hosting service, specify the URL to the audio clip. In the web chat, the linked audio clip renders by using the embeddable player for the hosting service.

     > **Note:** Specify the URL that you would use to access the audio file in your browser (for example, `https://soundcloud.com/ibmresearch/fallen-star-amped`). You do not need to convert the URL to an embeddable form; the web chat does this automatically.

     You can embed audio hosted on the following services:

     - SoundCloud
     - Mixcloud
3. Optionally specify a title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the audio player.

> **Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

Adding a *Video* response

Add a *Video* response to display a how-to demonstration, promotional clip, or other video content. In the web chat, a video response renders as an embedded video player.

The *Video* response type is supported by the following channel integrations:

- Web chat
- SMS
- Slack
- Facebook
- WhatsApp

To add a *Video* response, complete the following steps:

1. In the **Assistant says** field, click the **Video** icon.
2. In the **Source URL** field, type the full URL to the hosted video:
   - To link directly to a video file, specify the URL to a file in any standard format such as MPEG or AVI. In the web chat, the linked video renders as an embedded video player.

     > **Note:** HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.

   - To link to a video hosted on a supported video hosting service, specify the URL to the video. In the web chat, the linked video renders by using the embeddable player for the hosting service.

     > **Note:** Specify the URL that you would use to view the video in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). You do not need to convert the URL to an embeddable form; the web chat does this automatically.

     You can embed videos that are hosted on the following services:

     - YouTube
     - Facebook
     - Vimeo
     - Twitch
     - Streamable
     - Wistia
     - Vidyard
3. Optionally specify a video title, description, and alt text in the fields provided. In the web chat integration, the title and description are displayed along with the video player.

   > **Note:** References to variables are not supported. Some integration channels ignore titles or descriptions.

4. If you want to scale the video to a specific display size, specify a number in the **Base height** field.

Adding an *iframe* response

Add an *iframe* response to embed content from another website directly inside the chat window as an HTML `iframe` element. An iframe response is useful if you want to enable customers to perform some interaction with an external service without leaving the chat. For example, you might use an *iframe* response to display the following examples within the web chat:

- An interactive map on Google Maps
- A survey that uses SurveyMonkey
- A form for making reservations through OpenTable
- A scheduling form that uses Calendly

In the web chat, there are two ways the iframe can be included:

- Like a preview card that describes the embedded content. Customers can click this card to display the frame and interact with the content.
- Inline, meaning within the conversation. This option is good for smaller pieces of iframe content.

The *iframe* response type is supported by the following channel integrations:

- Web chat
- Facebook

To add an *iframe* response type, complete the following steps:

1. In the **Assistant says** field, click the **iframe** icon ( ).
2. Add the full URL to the external content in the **iframe source** field.

   The URL must specify content that is embeddable in an HTML `iframe` element. Different sites have different restrictions for embedding content, and different processes for generating embeddable URLs. An embeddable URL is one that can be specified as the value of the `src` attribute of the `iframe` element.

   For example, to embed an interactive map that uses Google Maps, you can use the Google Maps Embed API. For more information, see The Maps Embed API overview. Other sites have different processes for creating embeddable content.

   For the technical details of using `Content-Security-Policy: frame-src` that gives you permission to embed the website content in your assistant, see CSP: frame-src.
3. Optionally add a descriptive title in the **Title** field.

   In the web chat, the title that you add is displayed in the preview card. The customer clicks the preview card to render the external content.

   > **Note:** If you do not specify a title, the web chat attempts to retrieve metadata from the specified URL and displays the content title per the specification in the source.

   > **Note:** References to variables are not supported.
4. To show the iframe in the flow of the conversation, rather than as a preview card, set **Display iframe inline** to **On**. You can also set the height of the iframe. The default is 180 pixels.

Technical details: `iframe` sandboxing

Content that is loaded in an iframe by the web chat is *sandboxed*, meaning that it restricts permissions that reduce security vulnerabilities. The web chat uses the `sandbox` attribute of the `iframe` element to grant only the following permissions:

Permission	Description
`allow-downloads`	Allows downloading files from the network, if the download is initiated by the user.
`allow-forms`	Allows submitting forms.
`allow-scripts`	Allows running scripts, but *not* opening pop-up windows.
`allow-same-origin`	Allows the content to access its own data storage (such as cookies), and allows limited access to JavaScript APIs.

> **Note:** A script that runs inside a sandboxed iframe cannot change any content outside the iframe, *if* the outer page and the iframe have different origins. Be careful if you use an *iframe* response to embed content that has the same origin as the page where your web chat widget is hosted. In this situation the embedded content can defeat the sandboxing and gain access to content outside the frame. For more information about this potential vulnerability, see the `sandbox` attribute documentation.

Technical details: `iframe` preview card

The `iframe` response type in web chat displays the **Preview card**, which includes an image, title, and description of the webpage that the user visits in the web chat.

To display an image, title, and description in the **Preview card**, the webpage needs the following `<meta>` tags inside the `<head>` tag:

```
$ <meta property="og:image" content="https://.../image.jpg" />

<meta property="og:image:url" content="https://.../image.jpg" />

<meta property="og:title" content="The webpage title" />
<meta property="og:description" content="The webpage description" />
```

These metadata properties specified come from The Open Graph Protocol.

Tip: The metadata is optional. The web chat displays a preview card with the webpage url and metadata, that the web chat fetched successfully.

Pause response

Use a *Pause* response to have your assistant wait for a specified interval before the next response. This pause allows time for a request to complete, or acts like a live agent who pauses between responses. The pause must be of any duration from `0` to `60` seconds, which can include decimals of a second to configure the pause up to `milliseconds`.

A *Pause* response is typically used in combination with other responses. By default, a typing indicator animation appears during the pause to simulate a live agent.

The *Pause* response type is supported by the following channel integrations:

- Web chat
- Facebook
- WhatsApp

Note: With the phone channel, you can add a pause by including the SSML `break` element in the assistant output. For more information, see the Text to Speech documentation.

To add a *Pause* response:

1. In the **Assistant says** field, click the **Pause** icon.
2. In the **Duration** field, enter the length of time for the pause in `seconds`.

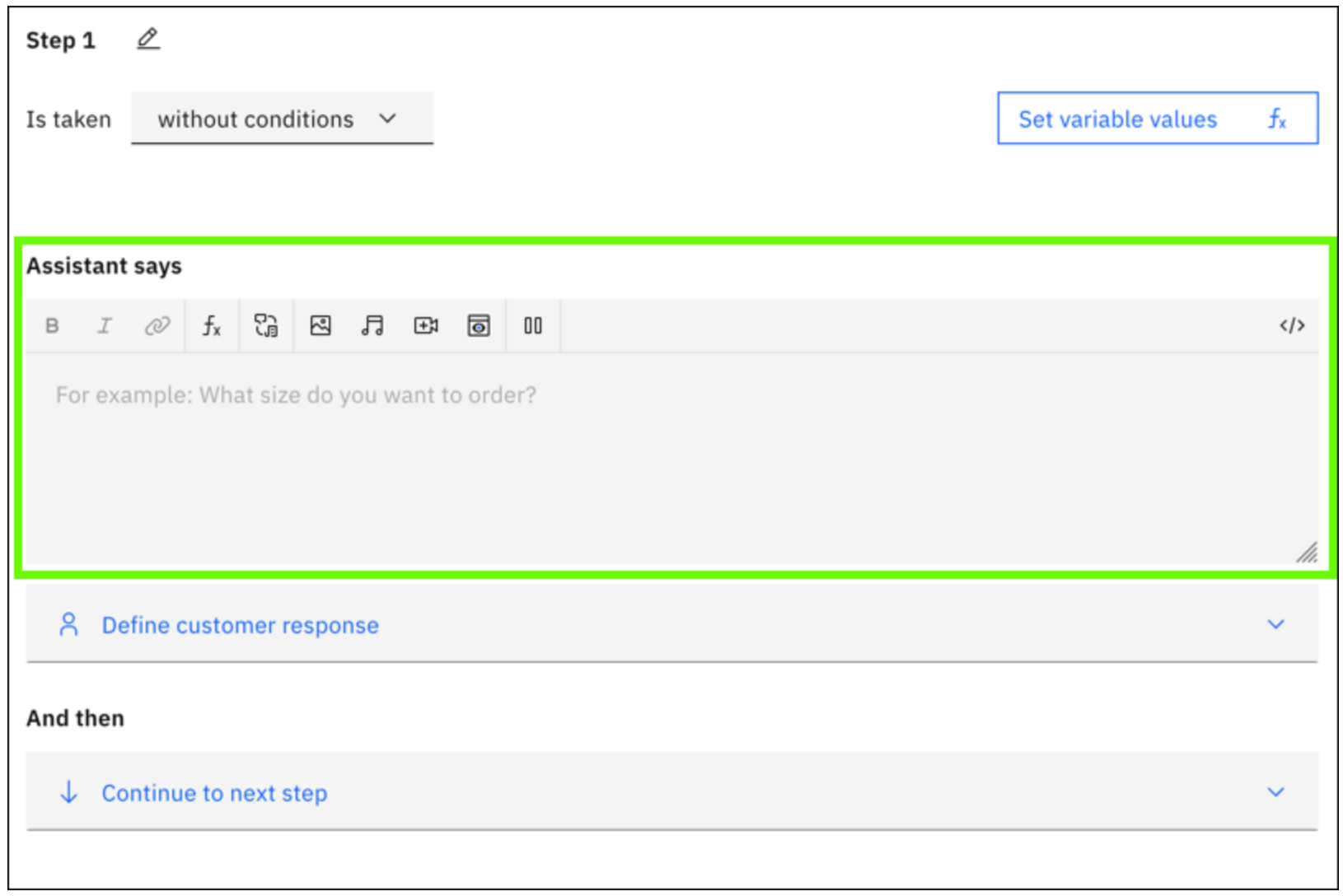
   The duration must not exceed `60 seconds`. In addition, you can enter the decimals of `second` in the **Duration** field to pause the response up to `milliseconds`. For example, if you enter `10.50 seconds`, the response pauses for `10500 milliseconds`. Customers are typically willing to wait about 8 seconds for someone to enter a response.
3. The **Typing indicator** is set to **On** by default. You can set this to **Off** if you want.

   Add another response type, such as a text response type, after the pause to clearly denote that the pause is over.

Collecting information from your customers

Many actions require multiple steps to collect all of the information that is required to complete the customer's request. When a step asks the customer for more information, the *customer response type* defines what type of response is expected.

In the step editor user interface, the middle portion of the step configuration defines the interaction between the assistant and the customer.



Step editor

The **Assistant says** field specifies the output that the assistant sends to the customer. If this output is a question that the user is expected to answer, that answer might be a number, a date, a name, or something else. You use the **Define customer response** field to specify what type of response is expected, based on the kind of information the assistant is asking for and how the customer is expected to specify it.

Choosing a response type

To choose the customer response type for a step, click **Define customer response** to expand the field. You can then select one of the following response types:

Response type	Description	Example input
Options	A list of predefined choices that customers can select from. At run time, the web chat integration shows an options response as a set of clickable buttons or as a drop-down list, depending on the number of choices.	`Small` `Medium` `Large`
Confirmation	A choice of either `Yes` or `No`. At run time, the web chat integration shows the `Yes` and `No` options as clickable buttons.	`Yes`, `No`
Regex	A text response that matches a specified pattern or format (such as an email address or telephone number).	
Number	A single generic number, specified either as numerals (100) or words (one hundred).	`100`, `one hundred`
Date	A single specific date or a range of dates.	`31 December 2021`, `12/31/2020`, `tomorrow`
Time	A single specific time or a range of time.	`5:00 PM`, `now`
Currency	An amount of money, including the unit.	`$25`, `500 yen`
Percent	A fractional numeric value that is expressed as a percentage.	`10%`, `50 percent`
Free text	Any arbitrary text response.	`123 Main Street`, `John Q. Smith`

Response types

Skipping steps, always asking steps, or never asking steps

Although a customer response is associated with a particular step, the assistant can recognize the required information at any point during the action. You can set a step to be skipped if its value is already provided in the user's input. If the value is specified after the step, the new value replaces the value that is specified in the step itself.

For example, if the customer's initial input was `I want to withdraw money from my checking account`, a step that asks the user to select a bank account is skipped because the customer already entered that information.

For steps that expect a customer response, you can decide whether to:

- Skip asking if the answer is mentioned in previous messages. This is the default for responses except *Confirmation* and *Free text*.
- Always ask for this information, regardless of previous messages. This is the default for *Confirmation* and *Free text*.
- Never ask. Collect information from previous messages.

Always ask

If your action asks for the same type of data in more than one step, use the **Always ask for this information** setting to prevent the assistant from making incorrect assumptions. For example, you might have an action in which one step asks for a hotel check-in date and another step asks for the check-out date. If you skip asking, the assistant can mistake the check-in date for the check-out date.

To require that a step is always used in the conversation with a customer:

1. In the customer response, click the **Settings** icon to open **Customer response settings**.

Customer response settings

2. Choose **Always ask for this information, regardless of previous messages** .
3. Click **Apply**.

Never ask

There might be some situations where you need a step to never ask a question because you anticipate redundant questions in the conversation.

To set that a step is never asked in the conversation with a customer:

1. In the customer response, click the **Settings** icon to open **Customer response settings**.
2. Choose **Never ask. Collect information from previous messages.**
3. Click **Apply**.

Example

This example explains when you might set a step to never ask for a response.

You might have an action that responds to requests to file an insurance claim. If you expect customers to typically make a request about a specific type of claim, such as auto, home, or medical, you might not want to ask another question about what type. They might say `I need to file an auto claim` or `I want to make a home claim.`

Although you still need a step that collects the answer about the type of claim, you might not want or need to ask that explicit question, especially if your assistant is used with the phone integration. Instead, you can create a step with the claim options, but set it to never ask.

This table shows how you might set up the steps. The last step is a catch-all in case the customer doesn't mention the claim type initially.

Step	Conditions	Assistant says	Customer response	Customer response setting	And then
1	None	What kind of claim?	Options: Automobile, Homeowner, Medical	Never ask	Continue to the next step

2	Step 1 is Automobile	None	Click here to file an automobile claim	Skip (default)	End the action
3	Step 1 is Homeowner	None	Click here to file a homeowner claim	Skip (default)	End the action
4	Step 1 is Medical	None	Click here to file a medical claim	Skip (default)	End the action
5	Step 1 is not defined (no claim type)	None	Click here to file an insurance claim	Skip (default)	End the action

Example using the never ask response setting

Protecting the privacy of the customer information

You can protect the privacy of the customer information in a step that you configure for the assistant. To hide confidential information in the conversation in a step, you can edit the settings and select the **Protect data collected at this step** checkbox. When you enable private-variable protection, the protected data is masked with asterisks in the conversation log. You can identify a conversation that includes a private variable in the preview page or preview panel with the **toggle-tip** icon ( ) that appears next to conversation.

Tip: To disable private variable protection, go to **Customer response settings** in the **Editor** tab inside an action step.

Note: The feature for masking the confidential customer information is available only for actions in assistants. If you're calling actions from a dialog, you can see the privacy setting in your action steps, but selecting the checkbox doesn't mask the customer information. In addition, this feature is not available for the assistants that are built in the classic experience of IBM® watsonx™ Assistant.

Note: For the stateful message API, the private variables are not included in the message response. For the stateless message API, the private variables are included in the message response with encryption.

Tip: Using variables to manage conversation information

Customer response types

The configuration information that you must provide varies by response type.

Options

An *options* response presents customers with a list of choices to select from. How these options are presented depends upon how your customers connect to the assistant. In the web chat integration, the options are shown as clickable buttons (for 4 or fewer options) or as a drop-down list (for more than 4 options).

There are two ways to create the list:

- Enter a list of options and synonyms
- Generate a dynamic list from a variable

Entering a list of options and synonyms

Enter each choice in the **Option** fields. You can click **Add synonyms** to enter variations of an option value that customers might type. You can enter multiple synonyms in a comma-separated list.

For example, you might define the following options and synonyms:

Option value	Synonyms
Blue	aqua, turquoise, navy
Red	burgundy, crimson, sangria

Green	lime, olive, forest

Options example

To select an option, customers can click an option button or list item, such as *Green*. Or they can type `Green` or one of its synonyms, such as `lime`.

Synonyms are useful for a response that might be skipped because they enable the assistant to recognize an option that the customer might have chosen before seeing the list. For example, if the customer's original input was `I want to order a large coffee`, a synonym would enable the assistant to recognize `large` as equivalent to the actual size `Grande`.

Tip: You can save this configuration for reuse in other steps. To save a customer response, click the **Save response for reuse** icon. For more information about saved customer responses, see Saving and reusing customer responses.

If you have a long list of options, such as all the states in the United States, you can choose to not show options in a list. This can be useful to prevent a phone integration from reading a long list of options to the customer.

To disable the list:

1. In the options customer response, click the **Settings** icon.
2. Click the **Present options to the customer in a list** toggle to off.

Dynamic list of options

Within the options customer response, you can use the **dynamic** setting to generate the list when you need to ask questions that are potentially different each time and for each customer. You need to set up a variable as the source of the options. For more information, see Dynamic options.

Confirmation

A *confirmation* response presents customers with the choices of either `Yes` or `No` as clickable buttons. Use this response type when the customer's response must be either Yes or No.

The following customer responses are recognized as `Yes`:

- `yeah`
- `yea`
- `yup`
- `sure`
- `positive`

The following customer responses are recognized as `No`:

- `not`
- `nope`
- `nay`
- `negative`

The default setting for *Confirmation* is **Always ask for this information**. For more information, see Skipping steps, always asking steps, or never asking steps.

Regex

A *regex* response collects a text string that matches a pattern that is expressed as a regular expression. Use this response to capture a value that must conform to a particular pattern or format, such as an email address or telephone number.

You can specify multiple regular expressions for a single response. For example, you might define multiple regex patterns that match part numbers from different vendors that use different formats. Input text for the response is recognized if it matches any of the regex patterns you specify.

To add a regex response:

1. In **Define customer response**, click **Regex**.
2. Click **Edit response**.
3. In the **Regular expression** field, select one of the predefined regular expressions, or click **Define custom regular expression** to write your own.

   To use a predefined regular expression, select one of the following:

- **Email**: An internet email address (for example, `user@example.com`).
- **Phone number**: A ten-digit US phone number (for example, `800-555-1212` or `(800) 555-1212`).
- **URL**: A correctly formatted URL for an online resource, optionally including the protocol (for example, `example.com` or `https://example.org/index.html`).

> **Tip:** For examples of other common patterns, see Example regex patterns.

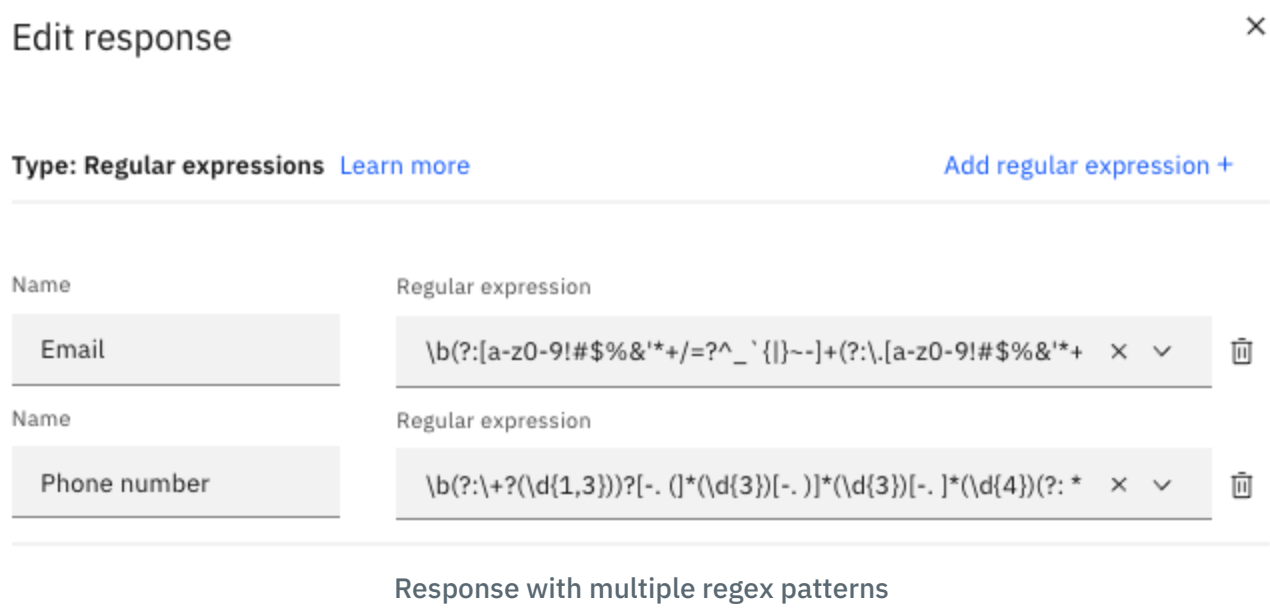
For more information on regular expression syntax, see Syntax.

Only English characters can be used in a regular expression. If you need to use other characters in a regular expression, you must represent those characters in Unicode.

> **Note:** watsonx Assistant uses the Google RE2 regular expression library to match regular expressions at run time. Regular expression syntax can vary between implementations, so ensure any regex patterns that you write conform to the RE2 syntax.

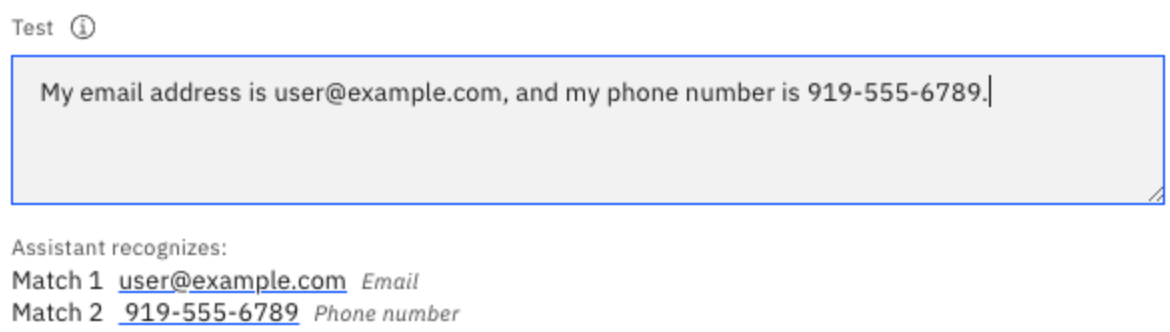
4. If you want to specify multiple regex patterns for the response, click **Add regular expression** to add another field in which you can select or define another regular expression.

   When you specify more than one regular expression, a **Name** field is displayed for each one. Use this field to give each regex pattern a unique name. You can use this name in subsequent step conditions to identify which regex pattern was matched.



Response with multiple regex patterns

5. Test your regular expression by typing example input in the **Test** field. If any text within your input matches the regex patterns you specified, the matching text is listed in the **Assistant recognizes:** field.



Match shown in regex test

> **Note:** The **Test** feature in the step editor uses a browser-based regex engine to find matches in your test input. At run time, the assistant uses a different regex engine that might have different results, especially with complex patterns. Before deploying your assistant in production, always use the assistant preview to test any step that uses a regex response.

6. Click **Apply** when you're finished editing.
7. By default, customers can't change topics when an utterance matches the pattern in a regex response. If you want a customer to be able to digress with a regex response, click the **Settings** icon, then enable the toggle **Allow customer to change topics before evaluating a regex response**. This gives the customer a chance to change topics before the assistant checks the regex pattern. For more information, see Allowing your customers to change the topic of the conversation.

> **Tip:** You can save your configured regex response for reuse in other steps. For more information, see Saving and reusing customer responses.

Example regex patterns

You can use the following regex patterns to recognize some common types of user input.

Description	Patterns
US passport number	`/^[A-PR-WY][1-9]\d\s?\d{4}[1-9]$/`
US bank routing number	`\b((0[0-9])|(1[0-2])|(2[1-9])|(3[0-2])|(6[1-9])|(7[0-2])|80)| ([0-9]{7})\b`
UPS tracking number	`/\b(1Z ?[0-9A-Z]{3} ?[0-9A-Z]{3} ?[0-9A-Z]{2} ?[0-9A-Z]{4} ?| [0-9A-Z]{3} ?[0-9A-Z]|[\dT]\d\d\d ?\d\d\d\d ?\d\d\d)\b/`
USPS tracking number	• `/(\b\d{30}\b)|(\b91\d+\b)|(\b\d{20}\b)/` • `/^E\D{1}\d{9}\D{2}$|^9\d{15,21}$/` • `/^91[0-9]+$/` • `/^[A-Za-z]{2}[0-9]+US$/`
FedEx tracking number	• `/(\b96\d{20}\b)|(\b\d{15}\b)|(\b\d{12}\b)/` • `/\b((98\d\d\d\d\d?\d\d\d\d|98\d\d) ?\d\d\d\d ?\d\d\d\d( ?\d\d\d)?)\b/` • `/^[0-9]{15}$/`

Example regex patterns

Number

A *number* response collects a single numeric value.

The customer can specify the number value in either numerals (`100`) or words (`one hundred`). Negative and decimal values are recognized.

Date

A *date* response collects a specific calendar date or a range of dates. The assistant can recognize dates that are expressed in various formats. Valid examples include:

- `Today`
- `Friday`
- `Now`
- `10/30/2020`
- `October 30th, 2020`
- `October 30th`

Time

A *time* response collects a single time or a range of times. The assistant can recognize times that are expressed in various formats. Valid examples include:

- `12:45PM`
- `10:30`
- `6am`
- `Now`
- `at 10`
- `from 5pm`
- `4 o'clock`
- `half past 4`

Currency

A *currency* response collects a currency value, including the amount and the unit. The assistant can recognize currency values that are expressed in various formats. Valid examples include:

- `$10.00`

- `20 cents`
- `five dollars`
- `500 yen`

Percent

A *percent* response collects a fractional value that is expressed as a percentage. The assistant can recognize a percentage that is written by using either the percent symbol (`%`) or the word `percent`). Valid examples include:

- `15%`
- `10.5 percent`

Free text

A *free text* response collects any arbitrary text string. Use this response for capturing any text, such as a name or address, or special instructions to be passed along. Valid examples include:

- `123 Main St.`
- `John Q. Smith`
- `Please add extra sauce`

The free text response type has these default settings:

- **Customer response collection behavior** is **Always ask for this information** and can't be modified. For more information, see Skipping steps, always asking steps, or never asking steps.
- **Change conversation topic** is disabled. If you want a customer to be able to digress and change topics while entering a free text answer, enable the toggle **Allow customer to change topics during a free text response** . For more information, see Allowing your customers to change the topic of the conversation.

You can enable watsonx.ai in your assistants to intelligently recognize the text strings in a free text response with multiple action variables. For more information about using watsonx.ai to gather information, see Information-gathering.

Saving and reusing customer responses

There might be some questions that your assistant needs to ask in multiple different steps and actions. For example, a banking assistant might support many different actions, each of which requires that the customer specifies an account number. A customer response might have a complex configuration (for example, it might have options with many synonyms). Instead of having to rebuild such a response over and over, you can save a customer response and reuse it wherever your assistant needs it.

Creating a saved customer response

To create a saved customer response:

1. In **Actions**, click **Saved responses**.
2. Click **New saved response**.
3. In the **Name** field, specify a descriptive name for the saved customer response configuration.
4. Configure the details of the response as required. A saved response can be created only as an options response type. For more information about this response type, see Customer response types.
5. Click **Save**. The saved customer response now appears on the **Saved responses** page.
6. In the **Type of response** field, choose Options or Regex.

⚠ **Important:** You can also edit or delete any existing saved customer response. Keep in mind that any changes that you make apply to all instances of the customer response in any step that uses it. If you delete a saved customer response, any steps that use that response become invalid and must be corrected to use a different response type.

Tip: For the options and regex customer response types, you can also create a saved customer response based on the customer response configuration within a step. If you configure a customer response in a step, click the **Save response for reuse** icon and specify a descriptive name for the saved customer response. (This isn't available if you use the dynamic setting for an options response.)

Uploading saved customer responses

If you have many saved customer responses, you can upload them from a comma-separated value (CSV) file than to define them one by one. If you are migrating entities from the classic experience to saved customer responses in watsonx Assistant, see Migrating intents and entities.

1. Collect the saved customer responses into a CSV file. Save the CSV file with UTF-8 encoding and no byte order mark (BOM).

   The required format for each line in the file is as follows:

   ```
   $ <savedResponse>,<value>,<synonyms>
   ```

   Where `<savedResponse>` is the name of a saved customer response, `<value>` is a value for the saved customer response, and `<synonyms>` is a comma-separated list of synonyms for that value. For example:

   ```
   $ genres,science fiction,sci-fi,SF
   genres,historical fiction,HF
   genres,young adult,YA
   genres,autobiography
   genres,biography
   genres,fantasy
   locations,Adams Street
   locations,Central
   locations,South End
   ```

   Uploading a CSV file also supports patterns. Any string that is wrapped with `/` is considered a pattern, as opposed to a synonym. For example:

   ```
   $ ContactInfo,localPhone,/(\d{3})-(\d{4})/
   ContactInfo,fullUSphone,/(\d{3})-(\d{3})-(\d{4})/
   ContactInfo,internationalPhone,/^(\(?\+?[0-9]*\)?)?[0-9_\- \(\)]*$/
   ContactInfo,email,/\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b/
   ContactInfo,website,/(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
   ```

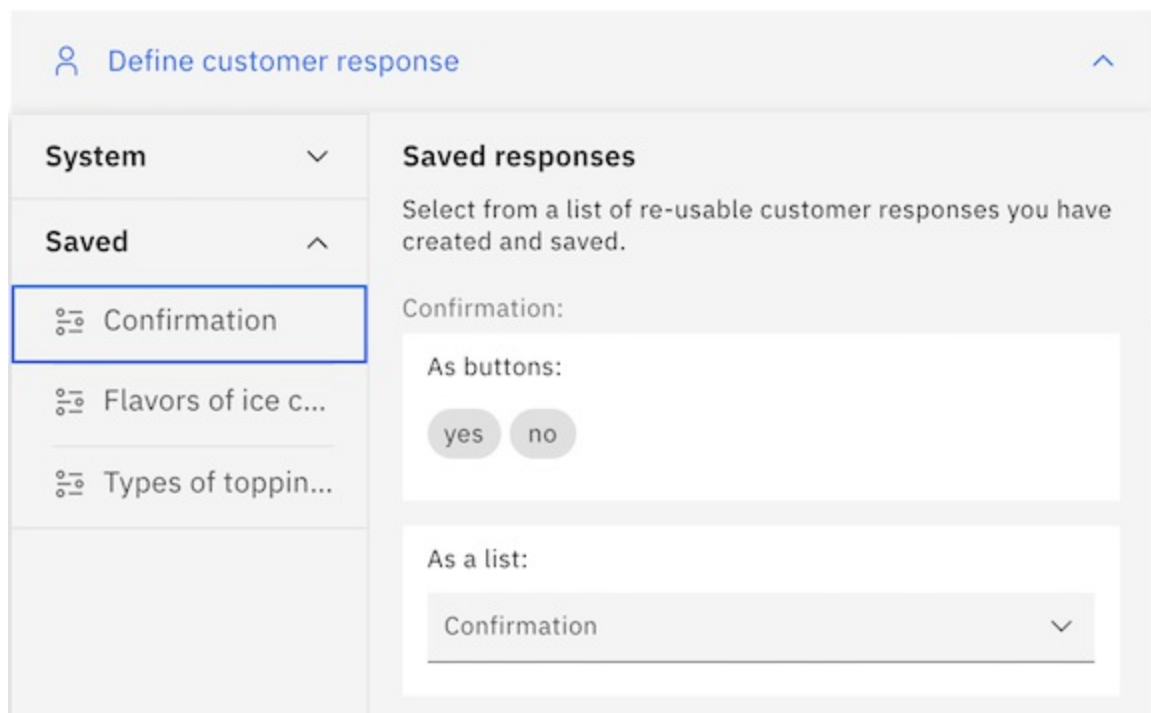
2. Go to the **Saved responses** page.
3. Click the **Upload** icon .
4. Select a file from your computer. The maximum CSV file size is 10 MB. If your CSV file is larger, consider splitting it into multiple files and uploading them separately.

   The file is validated and uploaded, and the system trains itself on the new data.

Using saved customer responses in steps

After you save a customer response, it becomes available as a response type for any step. To use a previously saved customer response in a step:

1. In the step editor, click **Define customer response**.
2. In the list of customer response types, click **Saved** to see the available saved customer responses.



Step editor

3. Click the saved response that you want to use.

To remove a saved response from a step, click the **Delete** icon. Removing a saved response from a step affects only the step that you are editing. It

does not delete the saved response or remove it from any other steps.

To edit a saved response from a step, click **Edit response**. Keep in mind that if you edit a saved response, your changes affect all steps that use the response. If you want to edit the response only for the step you are editing, click the **Unlink from saved response** icon. After you unlink a response, any edits you make affect only the step you are editing; they do not affect any other steps, nor are they applied to the saved response.

Tip: After you unlink a response, you cannot relink it. If you want to return to the saved response without your edits, delete the response, and then read the original saved response. If you want to make your edited version of the response available for reuse, save it as a new saved response.

Using variables to manage conversation information

When customers reply to your assistant, they share information about themselves and what they want. Your assistant remembers this information, and other information about a conversation, as *variables*. Your assistant can use variables to provide a more personalized and customized experience, and to get users quickly to the solutions they need.

Variables are a powerful tool that you can use to build a better assistant. Variables make possible all of the following benefits:

- **Personalization.** The best virtual assistant experiences are targeted and personalized for each customer. When an assistant greets a customer by saying "Hello, Frank! Welcome back," it tells Frank that it remembers his name and that it has talked to him before. By storing this kind of information in variables and then referencing them in your assistant's output, you can personalize the conversation and help your assistant seem more human.
- **Acceleration.** Over the course of a conversation, your customers answer questions and make choices. These customer responses are stored as variables, which your assistant can then use to guide a conversation. By choosing the right steps and not wasting your customers' time, you can get them as quickly as possible to the right solution.
- **Modularity.** Some information might be useful for many different purposes (for example, a customer's current account balance or contact information). Rather than retrieving or recalculating this information in multiple locations, you can do so once, by using a variable to store the result and then access it wherever you need it.
- **Privacy.** The privacy of the customer information is critical for all enterprises. In your assistant, you can mask the variables to protect the privacy of the shared information. When you mask a variable to make it private, the assistant hides the confidential information in the user input or assistant's responses with asterisks in the conversation logs.

Note: The feature for masking the confidential customer information is available only for actions in assistants. If you're calling actions from a dialog, the privacy setting is available in your action steps but won't mask the customer information.

When a user's session expires during a conversation in the web chat integration, the assistant loses the masked private variables. This is because web chat cannot copy the variables to a new session. For more information, see Copying session state.{: note}.

A variable is simply a named container for a piece of information; by referencing this container by name, your assistant can store or retrieve the information at run time. For example, a variable that is called *account_balance* might store your customer's current account balance, a value your assistant can update or retrieve as needed.

The data that is stored by a variable is characterized by the type of data that it contains, such as text, a numeric value, a date, or even a list of multiple values. The operations that you can perform with a variable vary depending on its data type.

Action variables and session variables

IBM® watsonx™ Assistant supports two categories of variables:

- **Action variables**: When a step collects information from the customer, the customer response is automatically stored in an *action variable*. You can think of action variables as short-term memory: they persist only during the current action.

  The name of an action variable is always the name of the step that defines the customer response. (You cannot change the name of an action variable.) For example, suppose you define a step that asks "When were you born?" and accepts a date value as a response. The customer response is automatically stored as an action variable called `When were you born?`, which you can then access from any subsequent step in the same action.

  You can make an action variable private by selecting the **Protect data collected at this step** checkbox in the customer response settings.

- **Session variables**: A value that is not necessarily tied to a particular action can be stored as a *session variable*. Session variables are long-term memory: they persist throughout the user's interaction with the assistant, and your assistant can reference them from any action.

  You can create a session variable to store the value from an action variable, if you want to keep the value available for other actions to use. You can also define a session variable based on another session variable, or by using a value defined in an expression. In addition to variables you create, watsonx Assistant provides a set of built-in session variables for global values like the current time and date.

> **Tip:** To hide the confidential customer information in the conversation logs, you can select the **Protect data stored in this variable** checkbox when you create or edit a session variable.

Session variables can help you to modularize your assistant because you can write a single action that collects information that is needed in multiple places. For example, you might have a greeting action that collects basic information about the customer and stores the responses in session variables, which any action can then access.

> **Note:** A session variable that you create persists only during a single session. At the end of the session, the variable's value is cleared. How long a session lasts depends upon how your customers access your assistant, and how your assistant is configured.

Creating a session variable

To add a session variable that can be accessed by any action:

1. From the **Actions** page, click **Variables > Created by you**. The list shows all session variables that you created for your assistant.
2. Click **New variable**.

   > **Tip:** You can also create a new session variable from the step editor. For more information, see Storing a value in a session variable.

3. In the **Name** field, type a name for the session variable.

   As you add the name, an ID is generated for you. Any spaces in the name are replaced with underscores (_) in the ID.

4. **Optional**: Add a type to set the response type of the variable. (For more information about response types, see Choosing a response type.)

   From this field, you can also select any of the saved responses that you created. For more information about saved responses, see Saving and reusing customer responses.

   In addition to the listed types, a variable can also be created as an array. To create an array variable, select **Any** as the type, and in the next step, define an initial value that uses the expression `[]` to represent an empty array.

5. **Optional**: Add an initial value to set the starting value for the variable at the beginning of each user session. For example, your customers can use an assistant to make purchases. You might initialize a *Payment due* variable with a starting value of 0, and then add to that value as the customer orders items.

   To specify a complex object or an array as the initial value, or to calculate the initial value based on other variables, you can write an expression. For more information about writing expressions, see Writing expressions.

6. **Optional**: Add a description.
7. **Optional**: Select the **Protect data stored in this variable** checkbox in the **Privacy** section if the session variable contains confidential information.
8. Click **Apply**.

Built-in variables

In addition to the variables you create, watsonx Assistant provides a set of built-in variables you can access from any action. At run time, these variables are automatically set with the appropriate values. For example, the *Current time* session variable always provides the current time in the user's time zone, at the time of the interaction with the customer.

To see these variables, click **Variables** on the **Actions** page.

- The **Set by assistant** page shows built-in session variables that are automatically provided for each assistant.
- The **Set by integration** page shows variables that are automatically provided by the integration your customer is using to connect to the assistant. (These variables are not set if no integration is connected.)

Set by assistant:

Variable name	Variable ID	Description	Examples
Digressed from	`digressed_from`	Last action before the customer digressed (or null if not digressed)	Pay bill
Now	`now`	The current date and time in the user's time zone.	`2021-08-11T11:28:02`

Current time	current_time	The current time in the user's time zone.	11:28:02
Current date	current_date	The current date in the user's time zone.	2021-08-11
Fallback reason	fallback_reason	The reason why a user is routed to the fallback action	Step validation failed - Agent requested - No action matches
No matches count	no_action_matches_count	Represents a count of customer's consecutive unrecognized input attempts	3
Session history IBM Cloud	session_history	The 24 most recent messages from the customer's conversation. For more information, see Session history	[{u : 'book a flight', n : true}, {a : 'sure! from where?'}]

Variables set by assistant

Set by integration:

Variable name	Variable ID	Description	Example
Timezone	timezone	The user's time zone as specified by the integration or API client. The default time zone (if not specified by the integration) is Coordinated Universal Time.	America/New_York
Locale	locale	The user's locale as set by the integration or API client. The locale can affect understanding and formatting of dates, times, and numbers.	en-gb
Channel Name	channel_name	The name of the channel that your user is interacting with.	Web chat

Variables set by integration

Storing a value in a session variable

Any action can store a value in a session variable so it is available to other actions. To store a value in a session variable:

1. From within a step, click **Set variable values**.
2. Click **Set new value**.
3. In the **Set** drop-down list, your choices are:

Choice	Description
Session variable	The session variable that you want to store the value in. The new value replaces any previous value that is stored.
Integration variable	The session variable that you want to store the value in. The new value replaces any previous value that is stored.
Expression	Write an expression directly without first picking a variable. For more information, see Writing expressions.
+ New session variable	You can create a new session variable, which is added to the list of session variables for the assistant. For more information, see Creating a session variable.

Set variable values

4. In the **To** drop-down list, the choices vary depending on the type of variable you're setting. Possible choices include:

Choice	Description
Scalar value by type	Set a specific value for each variable type. The choice varies depending on the variable type. For example, for a date variable, the choice is **Enter a date**, and you can use a date picker to set a date. Other choices appear for Boolean, confirmation, currency, date, free text, number, percentage, and time.

Expression	Write an expression to define the value for the session variable. For more information about expressions, see Writing expressions.
Action variables	Select an action variable to use the value of a customer response in another step. The choices that are listed match the type of variable that you want to set.
Session variables	Select another session variable to use its value. The choices that are listed match the type of variable that you want to set.
Assistant variables	Select a built-in system variable to use its value. The choices that are listed match the type of variable that you want to set.
Integration variables	If you are setting an integration variable, you can choose other integration variables as the value.

Set variable values

5. To set more variable values in the same step, click **Set new value**.

Using variables to manage conversation flow

One of the ways you can use variables is to choose the correct path through the conversation, based on customer responses and other values available at run time. You can do this by defining step conditions, which determine whether a specific step in an action is executed based on runtime conditions.

By defining a condition based on an action variable, you can control whether a step is executed based on the customer's response to a previous step. You can also build step conditions based on session variables, which can store information from other actions.

For more information about step conditions, see Defining step conditions.

Using variables to customize the conversation

You can also use variables in what your assistant says, dynamically referencing information that has been collected during the conversation. This is useful for confirming information the customer has provided (for example, `You want to transfer $153.14 to your checking account. Is that correct?`), and for simply personalizing the conversation to make it more human (`Hi, John. How can I help you today?`).

To reference a variable in what your assistant says:

1. In the **Assistant says** field, start typing the text for the response.
2. When you reach a point where you want to insert a reference to a variable, type a dollar sign (`$`) or click the *Insert a variable* icon ( ). A list appears showing the variables you can choose from.
3. Click a variable to add a reference to it in the text.

When you reference a variable, it appears using a default format in your assistant's response. The format of the variable might differ from the way the value is stored; for example, a date value of `2021-08-11` is formatted as `August 11, 2021` by default.

The default formats are as follows:

Type	Format	Examples
Options	As chosen by the user	`Yes` `No`
Number	Numerals only	`1000`
Date	Mmm DD, YYYY	`Jun 30, 2021`
Time	H:MM:SS AM	`5:15:00 PM`
Currency	Number only, no currency symbol	`20`
Percent	Number only, no percentage symbol	`20`

Free text	As entered by the user	`Please check that the apples aren't bruised`

Default formats for variables

Note: When building an assistant response that includes variables, you concatenate multiple parts (text strings and variables). A single response can consist of no more than 30 concatenated parts (for example, 15 variables along with 15 text strings).

Referencing expressions

If you need to reference a dynamic value that is calculated using an expression, you must first assign this value to a session variable. (For more information about how to do this, see Storing a value in a session variable.) You can then reference the session variable in the **Assistant says** field.

Note that the `<?...?>` syntax for referencing expressions in assistant output is not supported in actions.

Adding conditions to an action

IBM Cloud

An action condition is a Boolean test, based on some runtime value; the action runs only if the test evaluates as true. This test can be applied to any variable. By defining action conditions, you can do things such as control user access to actions or create date-specific actions.

For more information about variables, see Using variables to manage conversation information.

A basic action condition is expressed in the following form:

`If` `{variable}` `{operator}` `{value}`

where:

- `{variable}` is the name of a variable or an expression.
- `{operator}` is the type of test to apply to the variable value (for example, `is` or `is not`).
- `{value}` is the value to compare to the variable.

For example, an action condition might read:

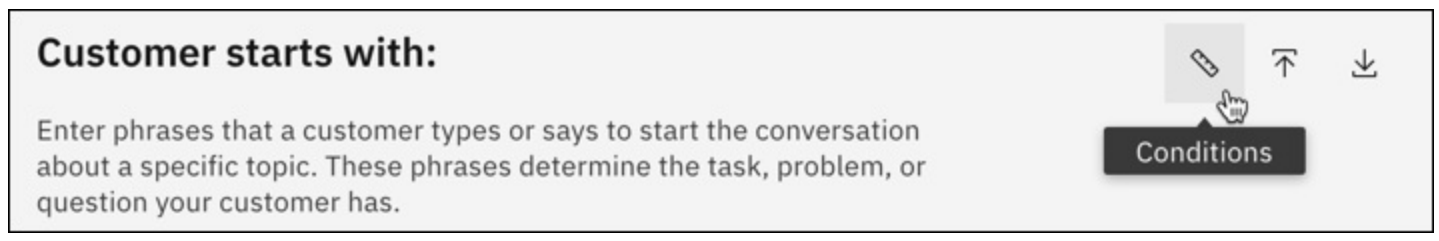
`If` `User category?` `is` `employee`

In this example, `User category` can be a list of employees at your organization. This condition evaluates as true if the user is an active employee. If false, you can control access so that former employees can't use the action.

Conditions can be grouped to construct complex tests.

To add an action condition:

1. In an action, click **Customer starts with**.
2. Click **Conditions**.



Conditions

3. Choose the variable for the condition. You can select:
   - An action variable with the customer response from a previous step in the action
   - A session variable that contains a value that is stored by any action
   - A built-in variable set by the assistant or by an integration

   Note: You can also define a complex condition by writing an expression that defines some other value. For more information about expressions, see Writing expressions.

4. Select the operator that represents the test that you want to perform on the variable (for example, `is` or `is not`). The available operators for a particular value depend upon its data type. For more information, see Operators.

5. Select the value that you want to evaluate the condition against. The values available depend upon the type of value that you are testing. For example, a variable that contains an options response can be tested against any of the defined options, and a date value can be tested against any date.

Conditions

6. To add more than one condition to an action, click **Add condition**.
7. To add another group of conditions, click **New condition group**.

   You can use groups to build complex action conditions. Each group is evaluated true or false as a whole, and then these results are evaluated together. For example, you might build an action that runs only if all conditions in group 1 are true *or* any condition in group 2 is true. (Groups function like parentheses in the Boolean conditions of many programming languages.)

   After you add a group, you can define one or more conditions in the new group. Between groups, choose **and** or **or** to indicate whether the conditions in both conditional groups or only one of them must be met for the step to be included in the conversational flow.

Operators

An operator specifies the kind of test that you are performing on a value in a condition. The specific operators available in a condition depend on the customer response type of the value, as shown in the following table.

Response type	Operators
• Options	• is • is not • is any of • is none of
• Regex	• is • is not

• Number • Currency • Percent	• is defined • is not defined • is equal to (==) • is not equal to (≠) • is less than (<) • is less than or equal to (<=) • is greater than (>) • is greater than or equal to (>=)
• Date	• is defined • is not defined • is on (also allows specific day of the week) • is not on • is before • is after • is on or before • is on or after
• Time	• is defined • is not defined • is at • is not at • is before • is after • is at or before • is at or after
• Free text	• is • is not • contains • does not contain • matches • does not match

Operators

Adding conditions to a step

An action represents a business process that helps customers answer their questions or solve their problems. Such a process must adapt to different specifics, based on information provided by customers or otherwise available at run time. For example, the steps for withdrawing money from a savings account might be slightly different from the steps for withdrawing for a checking account.

A step condition is a boolean test, based on some runtime value; the step executes only if the test evaluates as true. This test can be applied to any variable, such as an action variable containing the customer response from a previous step. By defining step conditions, you can create multiple pathways through an action based on different possible runtime values.

For more information about variables, see Using variables to manage conversation information.

A basic step condition is expressed in the following form:

`If` `{variable}` `{operator}` `{value}`

where:

- `{variable}` is the name of a variable or an expression.
- `{operator}` is the type of test to apply to the variable value (for example, `is` or `is not`).
- `{value}` is the value to compare to the variable.

For example, a step condition might read:

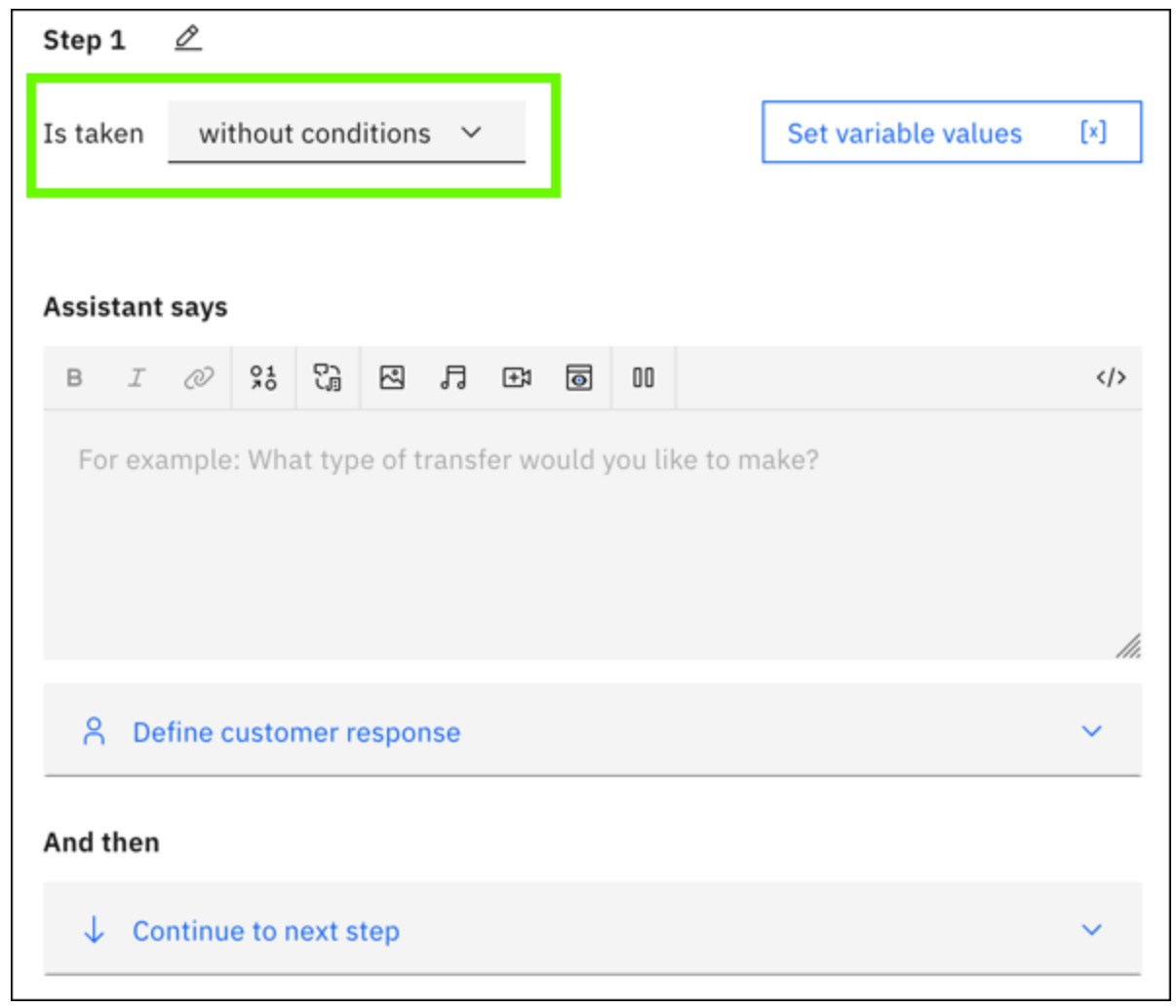
`If` `Withdraw from which account?` `is` `Checking`

This condition evaluates as true if the customer's response to the previous `Withdraw from which account?` step is `Checking`.

Conditions can be grouped together to construct complex tests.

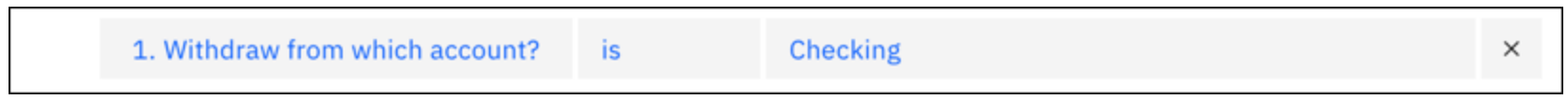
To add a step condition:

1. Open the step. Click the condition field at the beginning of the step:



Condition field

2. Select **with conditions** from the drop-down list. The **Conditions** section expands.
3. By default, a single condition group, containing a single condition, is automatically created based on the action variable stored by the most recent customer response.



Edit condition

You can click any part of the expression to edit it:

- Select the variable you want to test. You can select any of the following:
  - An action variable storing the customer response from a previous step in the action
  - A session variable containing a value stored by any action
  - A built-in variable set by the assistant or by an integration

  Note: You can also define a complex condition by writing an expression defining some other value. For more information about expressions, see Writing expressions.

- Select the operator representing the test you want to perform on the variable (for example, `is` or `is not`). The available operators for a particular value depend upon its data type. (For more information, see Operators.)
- Select the value you want to evaluate the test against. Again, the values available depend upon the type of value you are testing. For example, a variable containing an options response can be tested against any of the defined options, and a date value can be tested against any date.

4. To add more than one condition to a step, after adding a condition, click **New condition group**.

   One use case where using more than one condition is helpful is if you need to capture a value range. For example, maybe a requirement of opening a checking account is that the customer deposit at least $100 into the account at creation time. You might ask the customer if they want to transfer funds to the account, and if so, how much? To continue with the transfer, the transfer amount must be $100 or more, but cannot exceed $1000. You can add a step with the following conditions:

   - `How much to transfer?` `>` `99`
   - `How much to transfer?` `<` `1001`

   Specify whether all or any of the conditions must be met for the step to be included in the conversational flow.

5. To add another group of conditions, click **Add new group**.

   You can use groups to build complex step conditions. Each group is evaluated true or false as a whole, and then these results are evaluated together. For example, you might build a step that executes only if all conditions in group 1 are true *or* any condition in group 2 is true. (Groups function like parentheses in the boolean conditions of many programming languages.)

   After you add a group, you can define one or more conditions in the new group. Between groups, choose **and** or **or** to indicate whether the conditions in both conditional groups or only one of them must be met for the step to be included in the conversational flow.

Operators

An operator specifies the kind of test you are performing on a value in a condition. The specific operators available in a condition depend on the customer response type of the value, as shown in the following table.

Response type	Operators
• Options	• is • is not • is any of • is none of
• Regex	• is • is not
• Number • Currency • Percent	• is defined • is not defined • is equal to (==) • is not equal to (≠) • is less than (<) • is less than or equal to (<=) • is greater than (>) • is greater than or equal to (>=)
• Date	• is defined • is not defined • is on (also allows specific day of the week) • is not on • is before • is after • is on or before • is on or after

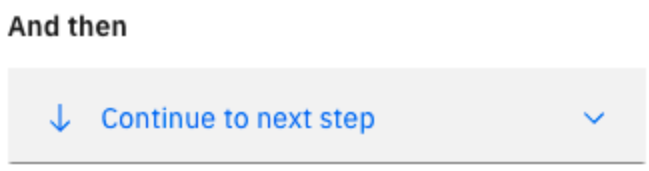
• Time	• is defined • is not defined • is at • is not at • is before • is after • is at or before • is at or after
• Free text	• is • is not • contains • does not contain • matches • does not match

Operators

Choosing what to do at the end of a step

By default, the steps in an action are executed in sequence from first to last. However, you can change this default in order to change what happens next. Used in combination with step conditions, this capability makes it possible for the conversation to follow many different flows depending on the customer's input.

To specify what happens when a step finishes, click **And then**.



Select an option from the drop-down list. The following options are available:

- Continue to next step
- Re-ask previous step(s)
- Go to a subaction
- Use an extension
- Search for the answer
- Connect to agent
- End the action

Continue to next step

This option processes the next step in the steps list. As always, the conditions for the next step are evaluated first to determine whether to show the step's response to the customer. This is the default selection.

Re-ask previous step(s)

This option repeats one or more steps that are listed earlier in the current action. These might be steps that the customer already completed, or steps that were skipped previously based on their step conditions.

You can use this option to handle situations where the customer has made a mistake and asks to go back to a previous point in the conversation. For example, you might include a confirmation step at the end of a process that asks the user whether the collected information is correct; if the user says no, you can return to the beginning of the process. This option is only available from a step that comes third or later in the steps list.

To repeat previous steps:

1. In the **And then** field, select **Re-ask previous step(s)**.

2. In the Settings window, click to select any previous steps you want to repeat. You can select any step that precedes the step you are editing.

   Note that only the selected steps will repeat, regardless of their **And then** settings. Therefore, if you want to repeat the entire action up to this point, you must select all of the previous steps.

   > **Tip:** The current step you are editing is automatically included in the list of steps to be repeated. To avoid an infinite loop, use step conditions to ensure that this step only executes when it is appropriate for previous steps to be repeated. For example, you might have a step that repeats previous steps only if the user answered `No` to a confirmation question; this way, if the user answers `Yes`, nothing is repeated and the action continues.

3. Click **Apply**.

Any session variable values that were defined based on choices that the customer made in the repeated steps are cleared and replaced with the new responses.

> **Note:** There is no option to jump to a later step. Instead of jumping directly to a later step, control the flow through the intervening steps with step conditions or skipping steps.

Go to a subaction

An action that is called from another action is referred to as a *subaction*. This option switches the conversation flow to another action. If you have an action flow that can be applied across multiple actions, you can use a subaction to build it once and then call it from each action that needs it. For example, as part of an action to place an order, you might call a subaction that enables a new customer to create an account.

To call a subaction:

1. In the **And then** field, select **Go to a subaction**.
2. In the Settings window, click the **Go to** field and select the action that you want to call.
3. If you do not want to continue with the current action, click **End this action after the other action is completed**. You might use this option in cases where the customer decides to do something different; in this case, you want the conversation flow to switch to the subaction and not return.

   By default, the assistant returns to the current action after the subaction completes. Any action variables or session variables that are defined in the subaction are accessible from subsequent steps in the calling action.

4. Click **Apply**.

Passing values to a subaction

Optionally, you can pass values to the subaction so the customer does not need to specify them again. For example, if your order-placement action collects the customer's name, you can then pass that information to the account-creation subaction. The step in the subaction that asks for the customer's name is skipped, and the value that is already specified is used instead.

> **Note:** External skill providers must implement the **Get Conversational Skill** endpoint to use this feature. For more information, see Get Conversational Skill.

To pass values to the subaction:

1. Click **Edit passed values**.
2. In the **Edit passed variables** window, click **Set new value**.
3. Select an action variable that you want to pass a value for. (The available action variables are based on the customer responses that are defined in the subaction.)
4. Select the value that you want to pass from the current action. You can select any available variable, or select **Expression** if you want to specify a different value.
5. Click **Apply**.

You can choose to hide or display the skill output that is stored as an assistant variable. Click **Response behavior** and select your choice.

Use an extension

You can call an extension that has been added to your assistant in order to interact with an external service. For example, you might use an extension to

interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions. For more information about calling an extension, see Calling an extension.

Search for the answer

Plus

This option indicates that the assistant can use a search integration to search an external data source for information that is related to the customer's question.

For more information about configuring search integration, see Adding search.

Configuring the responses

Perform the following procedure to configure how the responses are stored and displayed:

1. Enable conversational search to do post-processing.
2. In the **And then** field go to **Search for answer**.
3. Click **Edit settings** > **After Generation**.
4. You can configure settings in 3 different ways to store or display your responses after generation.
    - Only display the results and not store it.

      Select this option to display the response to the end user without storing it in an action variable.
    - Display and store the result.

      Select this option to store the response in an action variable and display it to your user.
    - Only store and do not display

      Select this option to store the response in an action variable only and do not display the response to the user.

When you are storing the response, you can choose to expand the citations to see the full search results. Use the **Search results** check box to include or exclude the full search results in the stored action variable. The exclude option is selected by default.

Conditioning the responses

You can view the action variable that will store the response under **Conditions**. You can see the list of conversational search responses in an action as **Conversational search (step n)**, where `n` is the number of the current step.

You can have multiple conversational searches in a single action. Click **Conversational search(step n)** to add a condition to the search status, to the search response text or to the action variable that will store the conversational search response.

Conversational search status

When storing the results of a conversational search, the search response will contain a search status corresponding to the results of the search. The possible values of this status field are:

- **Generated and below confidence threshold**
- **Generated and above confidence threshold**
- **Failed to generate**: no search results
- **Failed to generate**: search result confidence too low
- **Failed to generate**: unable to connect to search system or got error

Conversational search response

It is the text displayed as a response for your conversational search query.

Conversational search response type

A conversational search response type is available in actions where a conversational search step exists and the search results are saved in an action variable for post-processing. You can select the overall conversational search response to use in the response type. You can choose from the list of available conversational search responses.

Note: Even if streaming is enabled in an assistant, when using this new response type, the output will not be streamed.

You can print the results of the individual components of your search results by defining them in the **Variable values** in the rich text editor window. The printed results will not be formatted like they are in the response type, but is printed just as a string of text.

In the preview, go to **Variable values** > **Action variables** to view the action variables.

Note: For a conversational search step which stores the response, selecting end action after step will not be honored. If the action should end after a conversational search occurs, use the default after generation setting Display the response to the end user without storing it in an action variable.

Connect to agent

This option indicates that the assistant should transfer the conversation to a human agent. For more information, see Connecting to a live agent.

End the action

This option indicates that this action is complete. Any action variable values that were defined based on choices that the customer made during the action are reset.

This option can be applied to more than one step in a single action because an action can define more than one branch of a conversation, controlled by step conditions. For example, the open an account action might have one conversational flow for creating a checking account and a separate one for creating a savings account. Each branch might have its own final step. Identifying the final step helps analytical tools that follow a customer's progress through an action to identify the success or failure of the action.

Note: Using “End the action” in the last step functions like “Continue to next step”.

Adding search

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.

Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Administering your instance

IBM Watson® Discovery search integration setup

Plus | IBM Cloud Pak for Data | IBM Software Hub

The search integration searches for information from a data collection that you create by using the Discovery service.

Discovery is a service that crawls, converts, and normalizes your unstructured data. The product applies data analysis and cognitive intuition to enrich your data such that you can more easily find and retrieve meaningful information from it later. To read more about Discovery, see the product documentation.

Important: The search integration requires Discovery v2. For more information, see Getting the most from Discovery.

Typically, the type of data collection you add to Discovery and access from your assistant contains information that is owned by your company. This proprietary information can include FAQs, sales collateral, technical manuals, or papers written by subject matter experts. Mine this dense collection of proprietary information to find answers to customer questions quickly.

Watch a 4-minute video that provides an overview of the search integration:

View video: Search Integration: watsonx Assistant

Before you begin

Before you begin, you must:

- Set up a Discovery v2 instance on IBM Cloud or installed on IBM Cloud Pak for Data.
- Have at least a Plus plan Discovery service instance for IBM Cloud. Go to the Discovery page in the IBM Cloud catalog and create a Plus plan service instance.
- Use public endpoints. Private endpoints are currently not supported for watsonx Assistant or the classic experience.

Create the search integration or search skill

To create a search integration:

1. From the assistant where you want to add search, click **Integrations**.

   Note: You can add search if you are a user with a paid plan.

2. In the **Extensions** section, locate Search, click **Add**, then click **Confirm**.

If you are using the classic experience, add a search skill:

1. From the assistant where you want to add the skill, click **Add search skill**.
2. Take one of the following actions:
   - To create a new search skill, stay on the *Create skill* tab.
   - If you have created a search skill already, the *Add existing skill* tab is displayed, and you can click to add an existing skill.
3. Specify the details for the new skill:
   - **Name**: A name no more than 64 characters in length. A name is required.
   - **Description**: An optional description no more than 128 characters in length.

Connect to an existing Discovery instance

1. Choose the Discovery service instance that you want to extract information from.

   Note: If you see a warning that some of your Discovery service instances do not have credentials set, it means that you can access at least one instance that you never opened from the IBM Cloud dashboard. You must access a service instance for credentials to be created for it, and credentials must exist before watsonx Assistant can establish a connection to the Discovery service instance on your behalf. If you think a Discovery service instance is missing from the list, open the instance from the IBM Cloud® dashboard directly to generate credentials for it.

2. Indicate the data collection to use, by doing one of the following things:

   - Choose an existing project.

     You can click the *Open Discovery* icon to review the configuration of a project before you decide which one to use.

     Go to Configure the search.

   - If you do not have a project or do not want to use any of the projects that are listed, click **Create a new project** to add one. Follow the steps in Create a project.

     Note: Create a new project is not displayed if you reached the limits based on your Discovery service plan. See Discovery pricing plans for plan limit details.

Create a project

1. On the **OK, where is your data?** page, select your data source, then click **Next**. Example choices include Salesforce, SharePoint, Box, IBM Cloud Object Storage, web crawl, and upload data.
2. On the **Let's create a collection for your data** pages, enter the information on how to connect to your data and configure your collection. The information that you need to enter is different depending on the data source. For example, you need to enter your authentication credentials for services such as Salesforce, Sharepoint, and Box. For web crawl, you specify the website with your existing information.
3. Click **Finish**. Give Discovery a few minutes to start creating documents. You can use the **Manage collections** page within the project to see the progress.
4. Wait for the collection to be fully ingested, then click **Back to watsonx Assistant**.

Configure the search

1. On the watsonx Assistant search integration page, verify that the Discovery instance and project you want to use is selected, then click **Next**.
2. In the **Configure result content** section, review the Discovery fields and examples that are used in the search results shown to your customers. You can accept the defaults, or customize them as you want.

   The appropriate collection fields to extract data from vary depending on your collection's data source and how the data source was enriched. After you choose a data collection type, the collection field values are prepopulated with source fields that are considered most likely to contain useful information given the collection's data source type. However, you know your data better than anyone. You can change the source fields to ones that contain the best information to meet your needs.

   To learn more about the structure of the documents in your collection, including the names of fields that contain information you might want to extract, open the collection in Discovery, and then use the **Identity fields** and **Manage fields** tabs.

   Each search result can consist of the following sections:

   - **Title**: Search result title. Use the title, name, or similar type of field from the collection as the search result title.

     You must select something for the title or no search result response is displayed in the Facebook and Slack integrations.

   - **Body**: Search result description. Use an abstract, summary, or highlight field from the collection as the search result body.

     You must select something for the body or no search result response is displayed in the Facebook and Slack integrations.

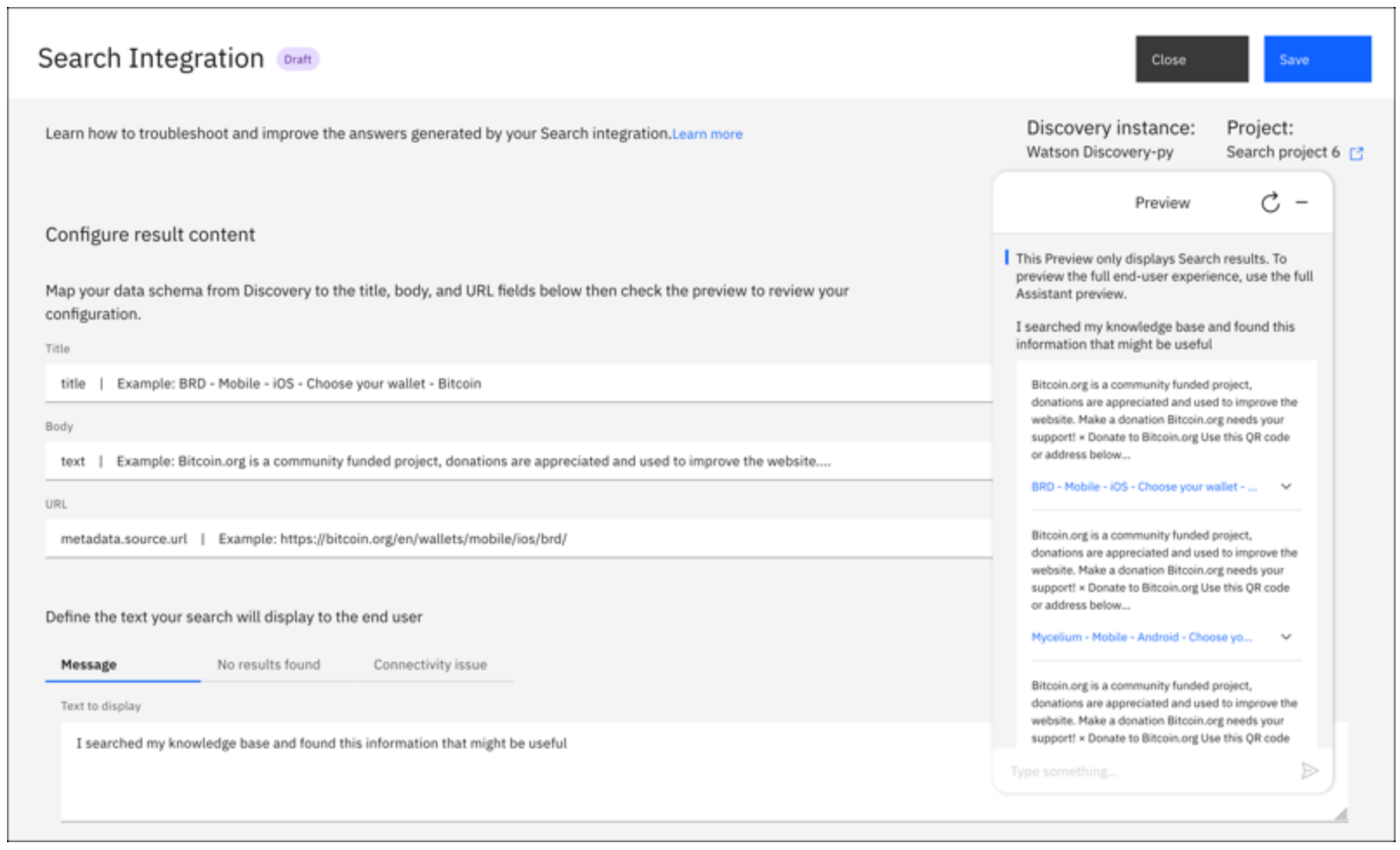
   - **URL**: This field can be populated with any footer content that you want to include at the end of the search result.

     For example, you might want to include a hypertext link to the original data object in its data source. Most online data sources provide self-referencing public URLs for objects in the store to support direct access. If you add a URL, it must be valid and reachable. If it is not, the Slack integration doesn't include the URL in its response, and the Facebook integration doesn't return any response.

     The Facebook and Slack integrations can successfully display the search result response when the URL field is empty.

> **Important:** You must use a field for at least one of the search results.

If no options are available from the drop-down fields, give Discovery more time to finish creating the collection. If the collection is not created, then your collection might not contain any documents or might have ingestion errors that you need to address first.



As you add field mappings, a preview of the search result is displayed with information from the corresponding fields of your data collection. This preview shows you what gets included in the search result response that is returned to users.

To get help with configuring the search, see Troubleshooting.

3. Use the **Message**, **No results found** and **Connectivity issue** tabs to customize different messages to share with users based on the successfulness of the search.

Tab	Scenario	Example message
Message	Search results are returned	I found this information that might be helpful:
No results found	No search results are found	I searched my knowledge base for information that might address your query, but did not find anything useful to share.
Connectivity issue	I was unable to complete the search for some reason	I might have information that could help address your query, but am unable to search my knowledge base at the moment.

Search result messages

4. Choose whether to enable **Emphasize the answer**.

> **Note:** This option is available only if your Discovery instance uses the v2 Discovery API.

When you enable this feature, the sentence that is determined by Discovery to be the exact answer to the customer's question is highlighted in the block of text that is displayed to the customer as the search result.

5. In the **Adjust result quantity** section, specify the number of results to return.

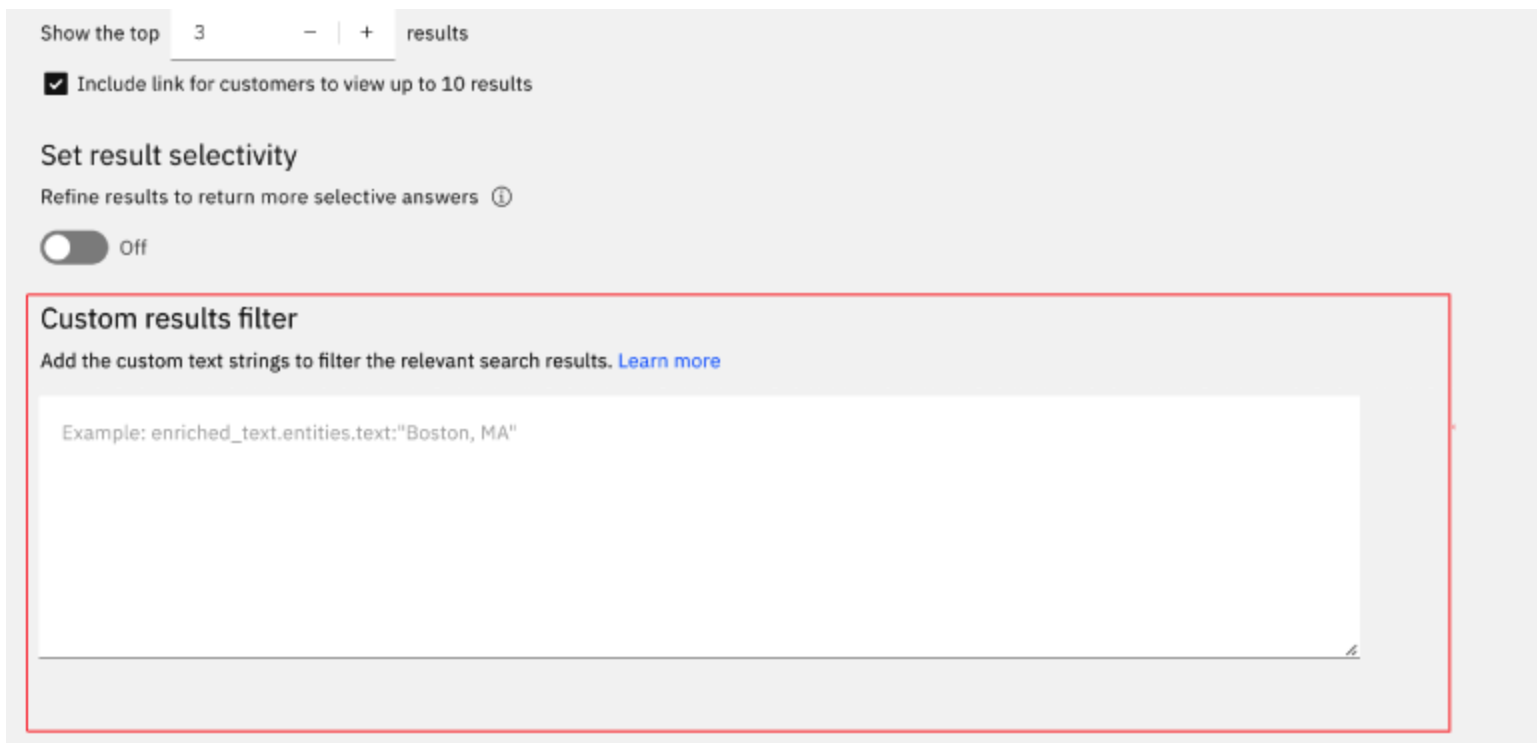
The top three results are returned automatically. You can choose to show fewer or more (up to 10) results in the response.

By default, customers can choose to see more results. If you don't want to give customers this choice, clear the **Include link for customers to view up to 10 results** checkbox.

6. In the **Set result selectivity** section, decide whether to be more selective with the answers that are returned. By increasing result selectivity, Search returns fewer but more accurate results. In most cases, Search is accurate enough that the default setting (off) is sufficient.

7. Use **Custom results filter** to add a filter for the custom text strings in the search integration. The **Custom results filter** field helps you define the

search results relevant for a topic, product, or text string. For example, if you define the **Custom results filter** field by sing `enriched_text.entities.text:"Boston, MA"`, the search responses for any query in the assistant are filtered to make it relevant to `"Boston, MA"` in the `enriched_text.entities.text` file.



8. Click **Preview**. Enter a test message to see the results that are returned when your configuration choices are applied to the search. Make adjustments as necessary.
9. Click **Create**.

Edit the search integration configuration

If you want to change the configuration of the search result card later, open the search integration again, and make edits. You do not need to save changes as you make them; they are automatically applied. When you are happy with the search results, click **Save** to finish configuring the search integration.

If you decide you want to connect to a different Discovery service instance or project, open the search integration and click **Edit Discovery Settings**. You can choose either a new project from the same instance, or a new instance and project.

Configure your assistant to use Discovery search

After you configure Discovery search integration, you must configure your assistant to use Discovery search when the customer response matches no action. For more information about updating **No matches** to use search, see Use search when no action matches.

Troubleshooting

Review this information for help with common tasks.

- **Creating a web crawl data collection**: Things to know when you create a web crawl data source:
  - To increase the number of documents that are available to the data collection, click add a URL group where you can list the URLs for pages that you want to crawl but that are not linked to from the initial seed URL.
  - To decrease the number of documents that are available to the data collection, specify a subdomain of the base URL. Or, in the web crawl settings, limit the number of hops from the original page. You can specify subdomains to explicitly exclude from the crawl also.
  - If no documents are listed after a few minutes and a page refresh, then make sure that the content you want to ingest is available from the URL's page source. Some web page content is dynamically generated and therefore cannot be crawled.
- **Configuring search results for uploaded documents**: If you are using a collection of uploaded documents and cannot get the correct search results or the results are not concise enough, consider using *Smart Document Understanding* when you create the data collection.

  You can annotate documents based on text formatting. For example, you can teach Discovery that any text in 28-point bold font is a document title. If you apply this information to the collection when you ingest it, you can later use the *title* field as the source for the title section of your search result.

  You can also use Smart Document Understanding to split up large documents into segments that are easier to search. For more information, see the Smart Document Understanding topic in the Discovery documentation.
- **My response text is surrounded by brackets**: If you notice that your response text is surrounded by brackets and quotation marks (`["My response text"]`) when you test it from the Preview, for example, you might need to change the source field that you're using in the configuration. The unexpected formatting indicates that the value is stored in the source document as an array. Any field that you extract text from must contain a value with a String data type, not an Array data type. When the chat integration shows a response that is extracted from a field that stores the data as an array, it does a straight conversion of the array value into a string, which produces a response that includes the array syntax.

For example, maybe the field in the source document contains an array with a single text value as its only array element:

```
"title": ["a single array element"]
```

The array value is converted by the watsonx Assistant into this string value:

```
"title": "[\"a single array element\"]"
```

As a result, the string is returned in this format in the chat; the surrounding brackets and quotation marks are displayed:

```
["a single array element"]
```

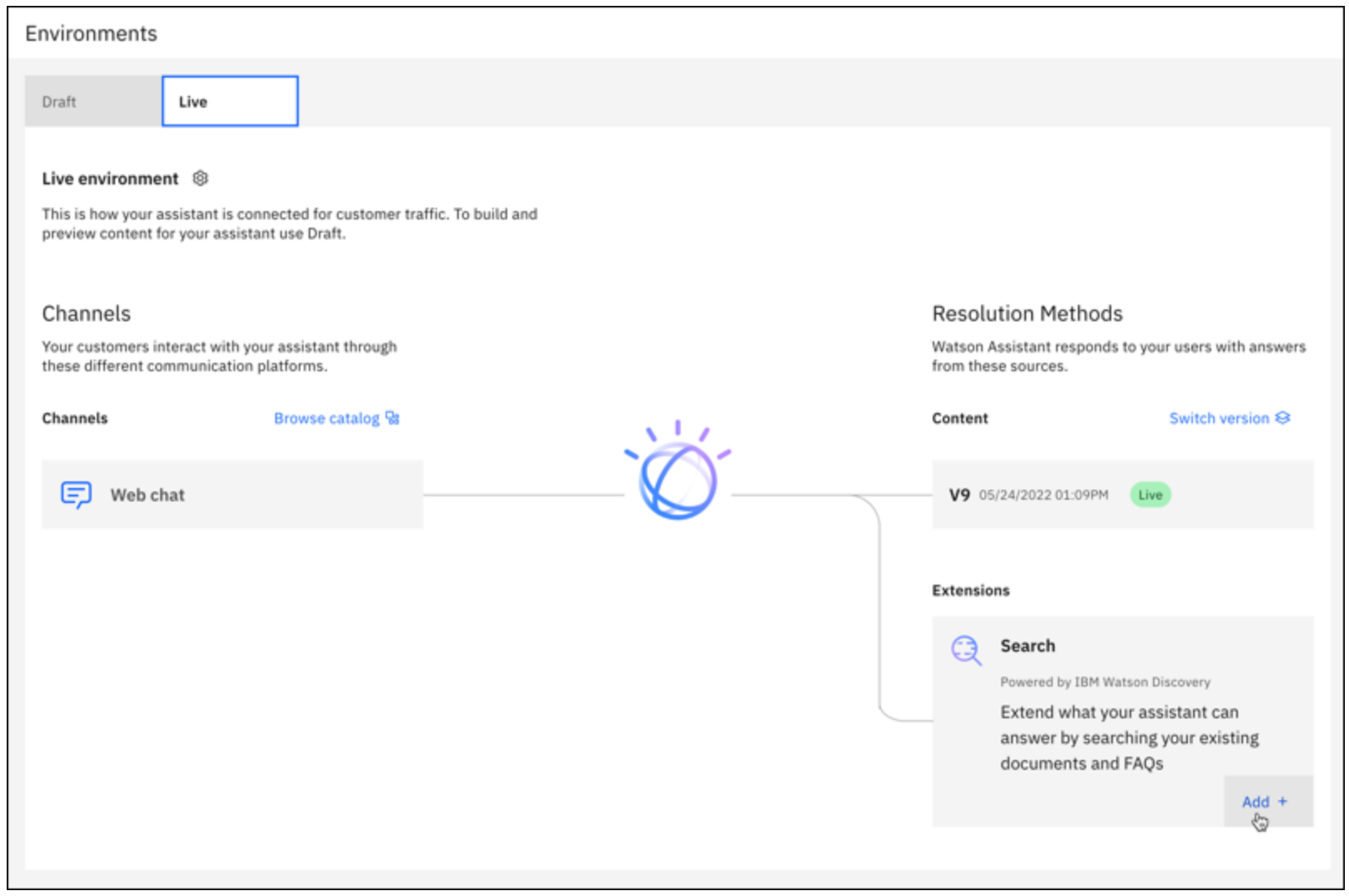
If you see this happening, consider choosing a different collection field from which to extract search results.

Note: The Discovery document `highlight` field stores values in an array.

Next steps

After you add the search integration for the first time, it appears as a tile on the **Draft environment** page. Click the tile to see or edit the search configuration.

When you're ready, you can repeat the steps to add the search integration to your **Live environment** or to other environments, if you're using multiple environments.



Test the search integration

After you configure search, you can send test queries to see the search results that get returned from Discovery by using the Preview page.

To test the full experience that customers have when they ask questions that are either answered by the action or trigger a search, use the *Preview* for your assistant.

Test the search skill in the classic experience

If you are using a search skill in the classic experience, you can send test queries to see the search results that get returned from Discovery by using the Preview pane of the search skill.

To test the full experience that customers will have when they ask questions that are either answered by the dialog or trigger a search, use the *Preview* button for your assistant.

> ⚠ **Important:** You cannot test the full end-to-end user experience from the dialog "Try it out" pane. The search skill is configured separately and attached to an assistant. The dialog skill has no way of knowing the details of the search, and therefore cannot show search results in its "Try it out" pane.

Configure at least one integration channel to test the search skill. In the channel, enter queries that trigger the search. If you initiate any type of search from your dialog, test the dialog to ensure that the search is triggered as expected. If you are not using search response types, test that a search is triggered only when no existing dialog nodes can address the user input. And any time a search is triggered, ensure that it returns meaningful results.

Sending more requests to the search skill

If you want the dialog skill to respond less often and to send more queries to the search skill instead, you can configure the dialog to do so.

You must add both a dialog skill and search skill to your assistant for this approach to work.

Follow this procedure to make it less likely that the dialog will respond by resetting the confidence level threshold from the default setting of 0.2 to 0.5. Changing the confidence level threshold to 0.5 instructs your assistant to not respond with an answer from the dialog unless the assistant is more than 50% confident that the dialog can understand the user's intent and can address it.

1. From the *Dialog* page of your dialog skill, make sure that the last node in the dialog tree has an `anything_else` condition.

   Whenever this node is processed, the search skill is triggered.

2. Add a folder to the dialog. Position the folder before the first dialog node that you want to de-emphasize. Add the following condition to the folder:

   `intents[0].confidence > 0.5`

   This condition is applied to all of the nodes in the folder. The condition tells your assistant to process the nodes in the folder only if your assistant is at least 50% confident that it knows the user's intent.

3. Move any dialog nodes that you do not want your assistant to process often into the folder.

After changing the dialog, test the assistant to make sure the search skill is triggered as often as you want it to be.

An alternative approach is to teach the dialog about topics to ignore. To do so, you can add utterances that you want the assistant to send to the search skill immediately as test utterances in the dialog skill's "Try it out" pane. You can then select the **Mark as irrelevant** option within the "Try it out" pane to teach the dialog not to respond to this utterance or others like it. For more information, see Teaching your assistant about topics to ignore.

Disabling the search skill

You can disable the search skill from being triggered.

You might want to do so temporarily, while you are setting up the integration. Or you might want to only ever trigger a search for specific user queries that you can identify within the dialog, and use a search skill response type to answer.

To prevent the search skill from being triggered, complete the following steps:

1. From the **Assistants** page, click the menu for your assistant, and then choose **Settings**.
2. Open the *Search skill* page, and then set the switch to **Disabled**.

Elasticsearch search integration setup

Plus Enterprise IBM Cloud Pak for Data IBM Software Hub

Elasticsearch powers your assistants to perform different types of searches such as metric, structured, unstructured, and semantic with higher accuracy and relevance by leveraging enterprise content. The data analytics engine in Elasticsearch expands the scope of search integration to larger datasets in assistants. In addition to this integration, you can enable conversational search for Elasticsearch in your assistant that helps to answer queries in a conversational manner.

> ⚠ **Important:** You can have only one search integration per environment. When you change the existing search integration to other integration types such as IBM Watson® Discovery or Milvus or Custom service, the settings of the existing search integration are overwritten.

Selecting Elasticsearch

To select Elasticsearch as the default search integration, use one of the following procedures:

- **Selecting Elasticsearch search integration from the Integrations page**

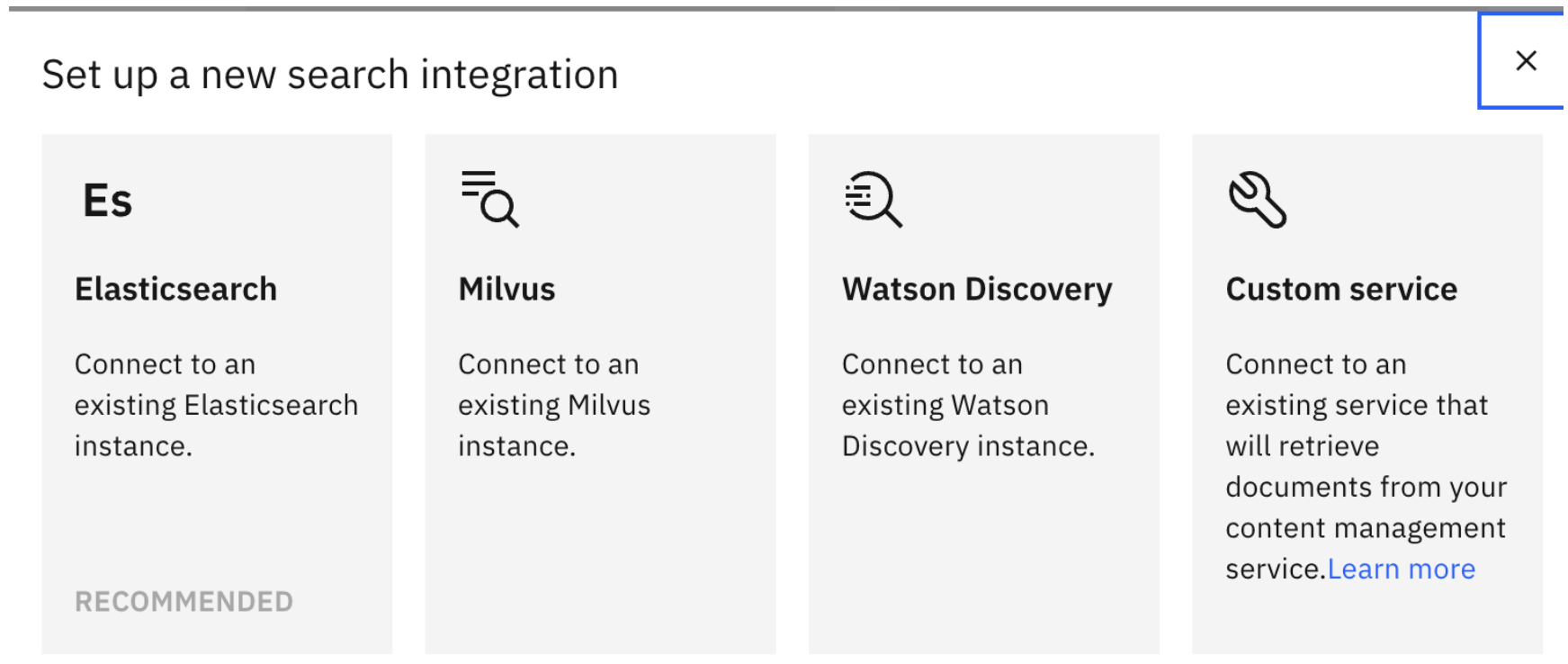
  1. After you create a watsonx Assistant instance, go to **Home** > **Integrations**.
  2. Click **Open** inside the **Search tile** to view the **Open Search** window.
  3. In the **Open Search** window, select the `Draft` option in the dropdown if you want to set up Elasticsearch in your assistant's draft environment. If you want to set up Elasticsearch in your assistant's live environment, select the `Live` option in the dropdown.
  4. In the **Edit an existing new search integration window** , select the **Elasticsearch** tile.

- **Selecting Elasticsearch search integration from the Environments page**

  1. After you create a watsonx Assistant instance, go to **Home** > **Environments**.
  2. Select the **Draft** tab if you want to set up Elasticsearch in the draft environment. If you want to set up Elasticsearch in the live environment, select the **Live** tab.
  3. In the **Resolution methods** section, click **Add** inside the **Search** tile under **Extensions** if you want to add an Elasticsearch search integration.

     > ☑ **Tip:** If you already added the Elasticsearch search integration, you see the **Open** button instead of **Add** inside the **Search** tile under **Extensions**.

  4. In the **Set up a new search extension** window, select the **Elasticsearch** tile.

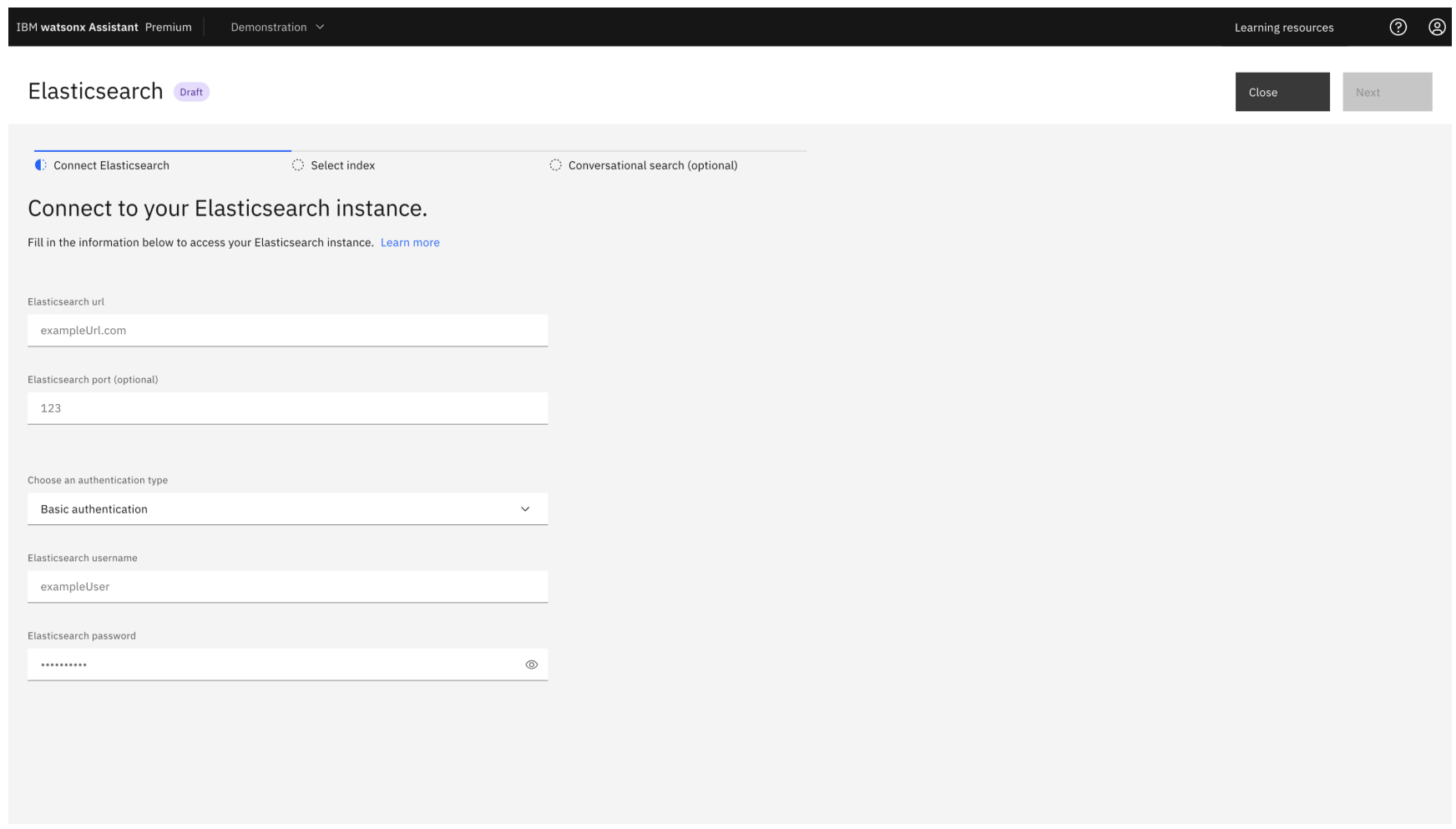


Select Elasticsearch

Setting up Elasticsearch

To set up Elasticsearch on your assistant, use the following procedure:

1. Provide the following fields to enable your assistant to connect to your Elasticsearch instance:
    - **Elasticsearch url**
    - **Elasticsearch port (optional)**
    - **Choose an authentication type**
        - If you select `Basic authentication`, you must provide an **Elasticsearch username** and an **Elasticsearch password**.
        - If you select `API key`, you must provide **Elasticsearch API key**.



Connect to Elasticsearch

2. Click **Next** to go to the **Select an index** section. **Select an index** has two options:
    - To use an existing index, select **Use my index**.
    - To create a new index, select **Upload documents to a new index in your Elasticsearch instance** .

Using an existing index

1. In the **Select index** section, click **Use my index** to connect to an existing Elasticsearch index. The **Use my index** option is selected as default in your

Elasticsearch set up.

2. In the **Use my index** option, type the Elasticsearch index name.
3. Click **Next** to go to the **Enable conversational search (optional)** section.
4. Conversational search is available only in the Plus and Enterprise plans of watsonx Assistant. In the **Enable conversational search (optional)** section, switch the **Conversational Search** toggle to `on` if you want to activate conversational search. If you don't want to activate conversational search, switch the toggle to `off`. For more information about conversational search, see conversational search.
5. Click **Save** and then **Close**.

Uploading documents to a new index Beta

Before you upload documents, your Elasticsearch instance must have the following prerequisites:

- Elasticsearch 8.8 or above.
- A paid or trial subscription of the Elasticsearch instance such as the Platinum Edition of IBM Cloud Databases for Elasticsearch or the Platinum or Enterprise subscription that is offered by Elastic.co.
- A Machine Learning (ML) node with a minimum of 4 GB memory to deploy the ELSER model. For more information about the ELSER requirements, see ELSER requirements.
- The documents that you upload must be in English.

If your Elasticsearch instance does not have the prerequisites for uploading documents, you see the `Requirements not met` error message.

> **Note:** If you experience a delay or failure in uploading documents even after meeting the prerequisites, you can consider scaling the inference performance of the ELSER model deployment by setting up parameters such as `number_of_allocations` and `threads_per_allocation`. For more information about scaling the inference performance, see Start trained model deployment API.

To upload documents to a new index, use the following procedure:

1. In the **Select index** section of the **Elasticsearch** window, click **Upload documents to a new index in your Elasticsearch instance** .

   watsonx Assistant passes the uploaded documents to your Elasticsearch instance for storage, chunking, and indexing.

2. In the **Configure result content** section, provide the following fields to map the title, body, and URL to the search response:

   - **Title**

     Search result title. Use the title, name, or similar type of field from the collection as the search result title.

     You must select something for the title or no search result response is displayed in the Facebook and Slack integrations.

   - **Body**

     Search result description. Use an abstract, summary, or highlight field from the collection as the search result body.

     You must select something for the body or no search result response is displayed in the Facebook and Slack integrations.

   - **URL**

     This field can be populated with any footer content that you want to include at the end of the search result.

   When you configure the query body in the **Advanced Elasticsearch Settings** to search the nested documents, you must ensure that the **Title**, **Body**, and **URL** are from the fields of the inner documents in your Elasticsearch index. For more information about using nested queries, see Elasticsearch nested query.

3. Expand the **Advanced Elasticsearch settings** section to see the following text boxes:

   - **Configure the filter array for Elasticsearch**

     You define the filter as an array of objects so that you can create filters to arrange the content per the query body. For more information, see Configuring the custom filters.

   - **Configure the query body for Elasticsearch**

     The query body is used to manipulate the user requests into a format that the search expects. It controls the query forms, search fields, filters, and query size. In the REST API, the query body is an object that represents the `POST` body for the `_search` request to Elasticsearch. THe

query body has a `"$QUERY"` token to represent the customer's query, and a `"$FILTER"` token to represent the array of filters that are defined either in the search settings or at the step level.

By default, Elasticsearch integration uses keyword search. But you can configure the query body in the `Advanced Elasticsearch settings` to enable more advanced search techniques such as:

- Semantic search with ELSER
- KNN dense vector search
- Using nested queries to search the nested documents
- Hybrid search
- Search on a semantic text field

For more information about using different types of query body examples, see Query body examples.

For more information about the Elasticsearch `_search` API request body, see Elasticsearch search API request body.

> ⚠ **Important:** You cannot customize the query body in the assistant with an existing Elasticsearch configuration.

4. Use the **Message**, **No results found** and **Connectivity issue** tabs to customize different messages to share with users based on the successfulness of the search.

Tab	Scenario	Example message
Message	Search results are returned	`I found this information that might be helpful:`
No results found	No search results are found	`I searched my knowledge base for information that might address your query, but did not find anything useful to share.`
Connectivity issue	I was unable to complete the search for some reason	`I might have information that could help address your query, but am unable to search my knowledge base at the moment.`

Search result messages

5. Switch the **Conversational Search** toggle to `on` if you want to activate conversational search. If you don't want to activate conversational search, switch the toggle to `off`.

   Conversational search is available only in the Plus and Enterprise plans of watsonx Assistant. If you switch the **Conversational Search** toggle to `on`, you can see the citation titles in your assistant responses. For more information about conversational search, see conversational search.

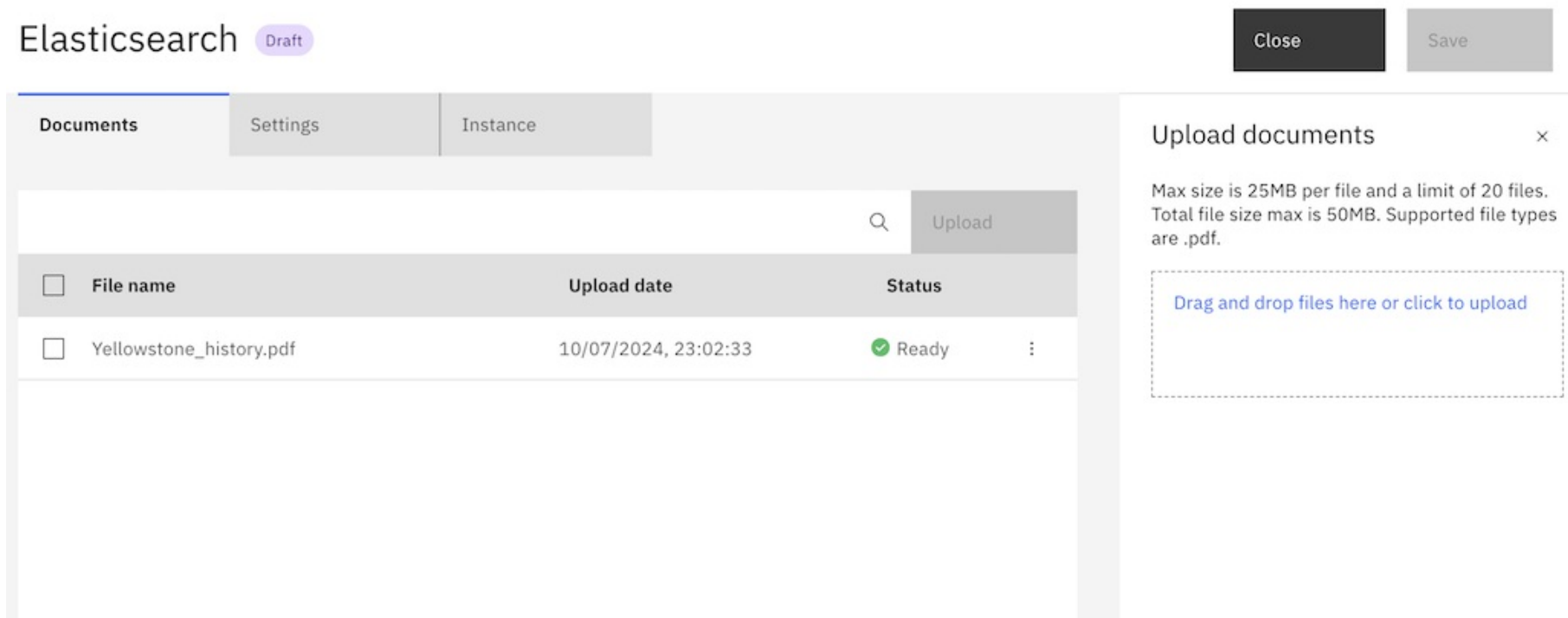
6. Click **Save** to save your settings.

7. Click the **Documents** tab in the **Elasticsearch** window.

> ☑ **Tip:** The **Documents** tab is enabled only if you select **Upload documents to a new index in your Elasticsearch instance** option.

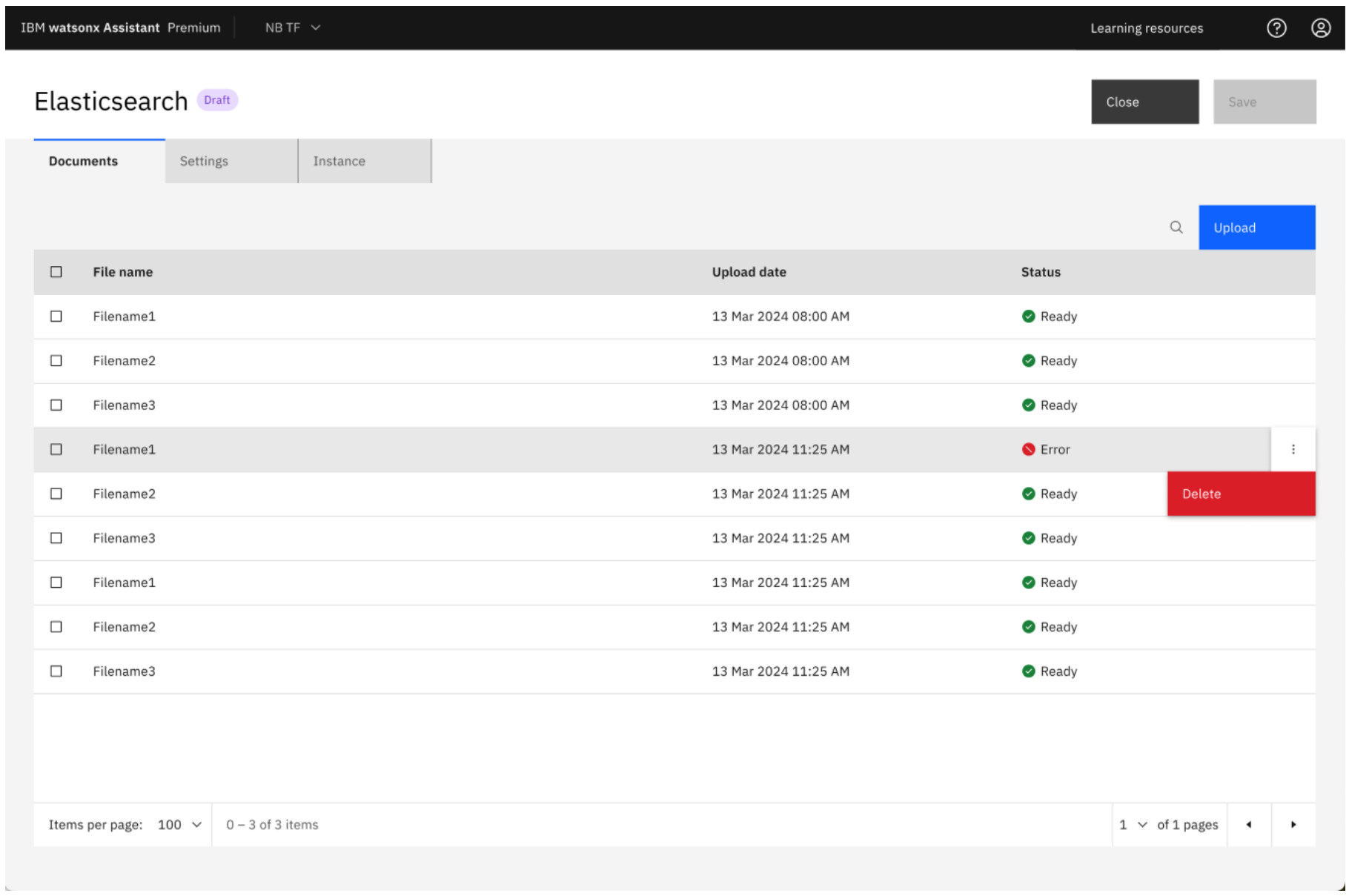
8. Click **Upload** button. In the **Upload documents** section, you can drag your files or do a single click to upload documents directly to your assistant.

> ⚠ **Important:** You can upload up to 20 documents at a time. Each document file size must not exceed 25 KB. The total size of all documents must not exceed 50 MB.



Elasticsearch upload document

9. After you upload the documents, you can see the upload status of your documents in a table in the **Elasticsearch** window.
10. Status `Ready` indicates that your files are available for search.
11. If the status indicates `Error`, you can delete the file by clicking the three dots next to the `Error` and click `Delete`.



Elasticsearch upload error

12. Skip this step if you do not want to change Elasticsearch instance credentials. If you want to change the Elasticsearch instance credentials, click the **Instance** tab, edit the credentials, and then click **Save**.
13. Click **Save** and then **Close** to end the Elasticsearch set up.

Configuring your assistant to use Elasticsearch

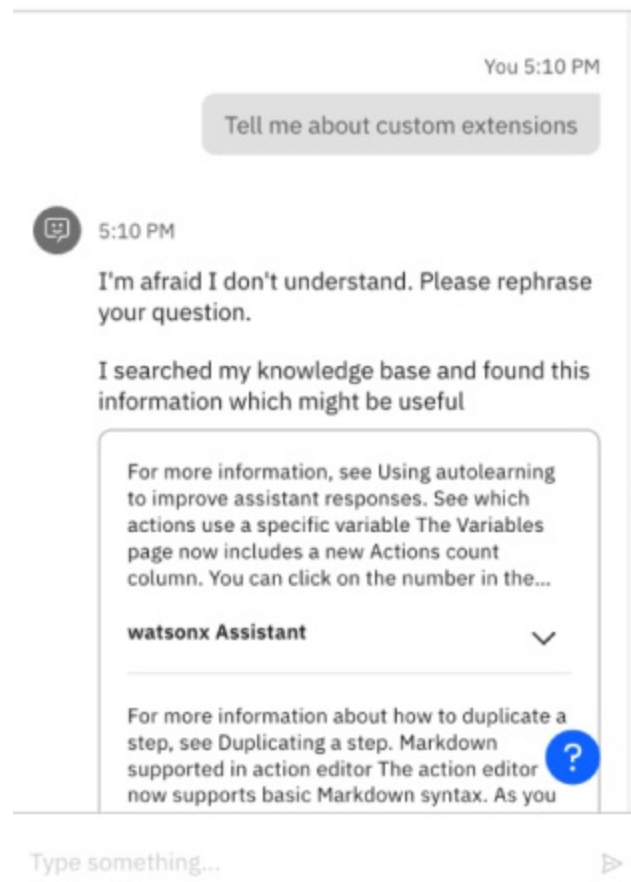
After you configure the Elasticsearch search integration, you must configure your assistant to use Elasticsearch when the customer response matches no action. For more information about updating **No matches** to use search, see Use search when no action matches.

Testing Elasticsearch

You can test search integration with Elasticsearch in actions preview, the preview page, or by using the preview link.

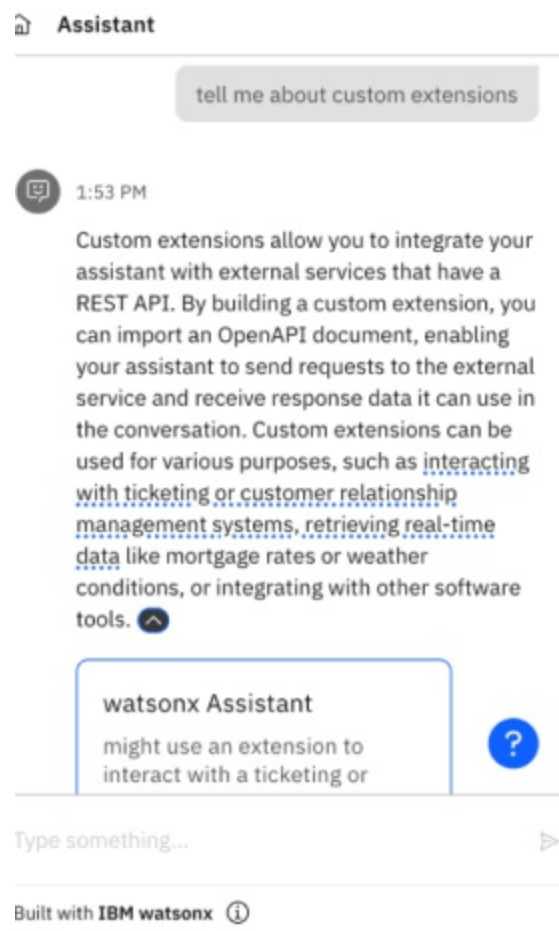
In this example, the user asks, `Tell me about a custom extension`.

Search results are pulled from your knowledge base when conversational search is `off`. The answer is, `I searched my knowledge base and found this information which might be useful`.



Conversational search off

A text-based reply from the best results in your knowledge base displays when conversational search is `on`.



Conversational search on

Milvus search integration setup

Plus Enterprise

Milvus is a vector database that you can use for handling large-scale datasets. For applications that require real-time search capabilities and numerous concurrent users, you can use Milvus, which has a distributed architecture, high performance, and flexible data model.

> ⚠ **Important:** You can have only one search integration per environment. When you change the existing search integration to another integration types such as IBM Watson® Discovery, Elasticsearch or Custom service, the settings of the existing search integration are overwritten.

Prerequisites for connecting Milvus to assistant

- You must have an active account on watsonx.data and watsonx.ai.

  To create an account on watsonx.data see, watsonx.data home page.

To create an account on watsonx.ai, see watsonx.ai home page.

- You must provision a Milvus instance in watsonx.data.

  For more information on creating a Milvus instance in watsonx.data, see Adding Milvus service in watsonx.data.

  For more information on updating your Milvus authentication credentials, see Updating your Milvus authentication credentials.

  For more information on creating an API key, see API keys.

- You must configure watsonx.data Milvus in IBM watsonx.ai.

  To configure watsonx.data in IBM watsonx.ai, see Configuring watsonx.data in IBM watsonx.ai.

  For more details on creating a vector index, see Creating vector index.

  For more information about creating collections, see Creating collections.

Connecting Milvus to assistant

Integrating Milvus in assistant involves three platforms.

1. watsonx Assistant, where we build the integration.
2. watsonx.data, where we are provisioning the Milvus instance.
3. watsonx.ai, where we are building the data source called collections.

Selecting Milvus as a search integration in assistant

After you create a watsonx Assistant instance, you can select Milvus as the search integration by using one of the following procedures:

- **Selecting Milvus search integration from the Integrations page**

  1. Go to **Home** > **Integrations**.
  2. Scroll down to the **Extensions** section. In the **Search** tile, click **Add** to open the **Set up a new search integration** window.
  3. Select **Milvus** to open the **Connect your search provider** window.

- **Selecting Milvus search integration from the Environments page**

  1. Go to **Home** > **Environments**.
  2. In the **Resolution methods** section, go to the **Extensions** section.
  3. Click **Add** inside the **Search** tile to open the **Set up a new search integration** window.
  4. Select **Milvus** to open the **Connect your search provider** window.

- **Selecting Milvus search integration through Conversational search**

  1. In the **Home** page, scroll down to the **Assistant architecture** section.
  2. In the **Conversational search** node, click **Add** to open the **Set up a new search integration** window.

> **Note:** If Milvus is already added, the **Open** button is displayed. Else, **Add** is displayed. To set up custom service on the **Draft** or **Live** assistant environment, click **Open**.

Set up a new search integration

Elasticsearch

Connect to an existing Elasticsearch instance.

RECOMMENDED

Milvus

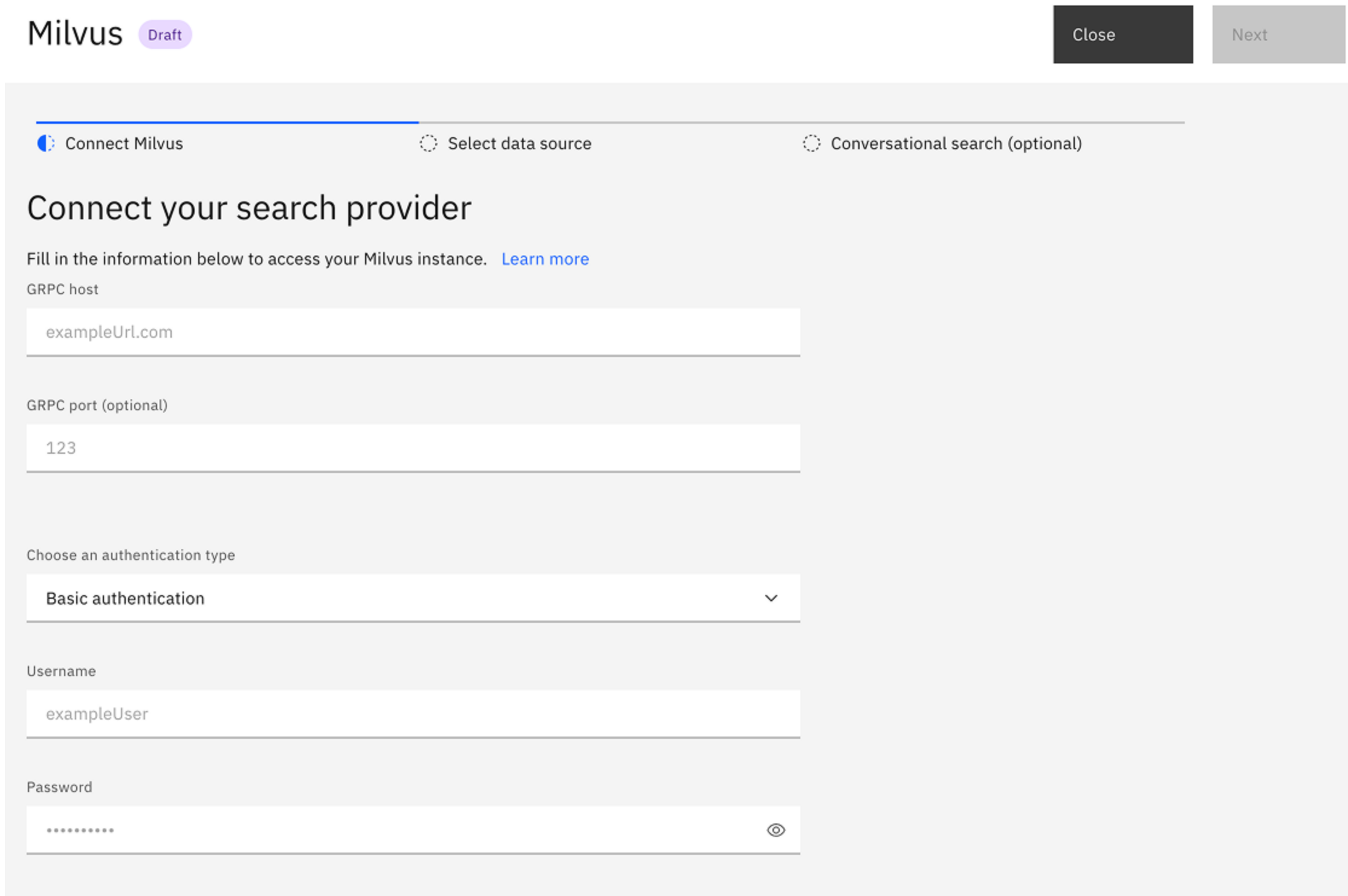
Connect to an existing Milvus instance.

Watson Discovery

Connect to an existing Watson Discovery instance.

Connecting to watsonx.data to set up Milvus

After you select Milvus as the search integration, use data on your Milvus instance in watsonx.data to set up Milvus on your assistant.

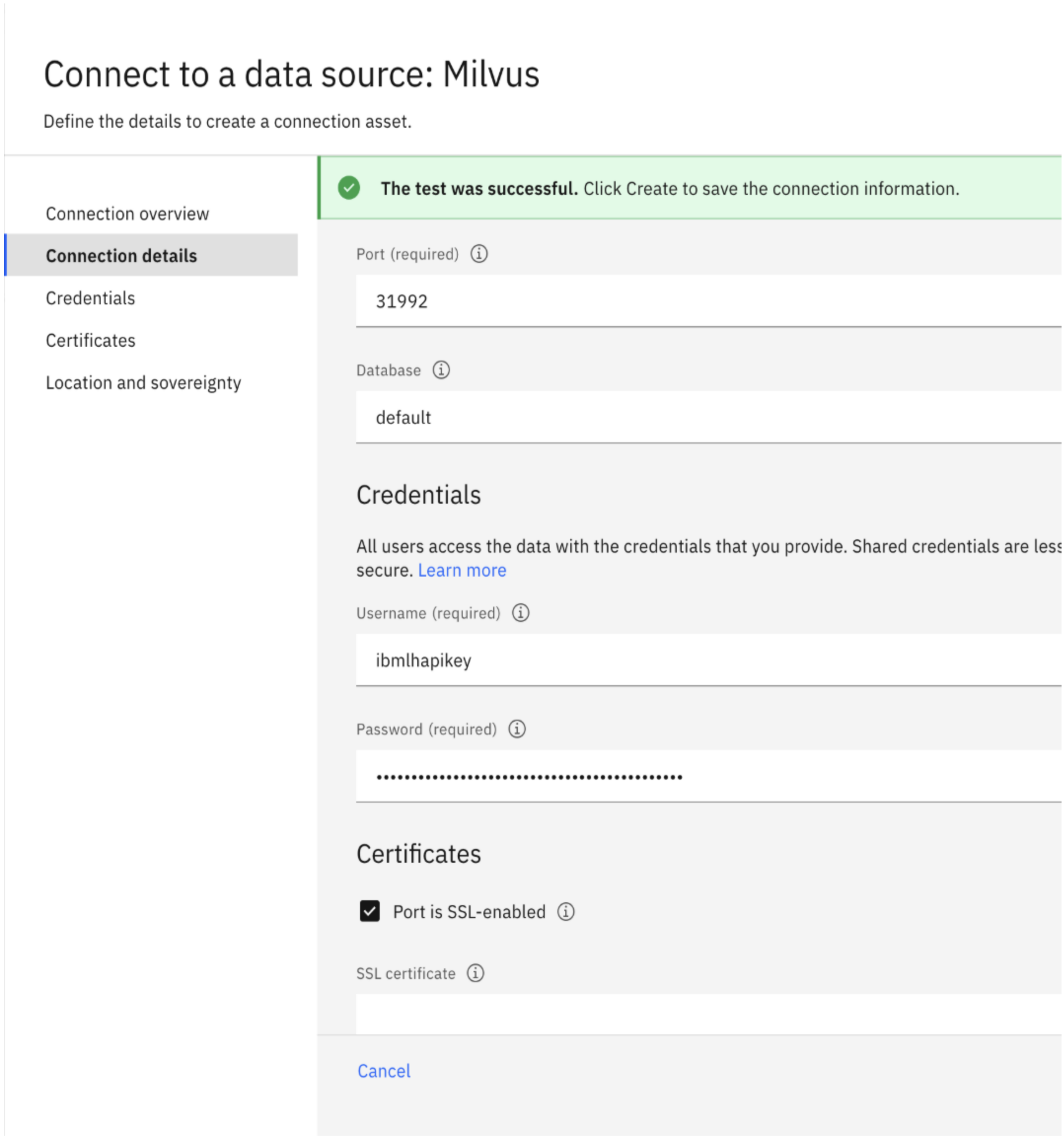


For more information on creating a Milvus instance in watsonx.data, see Prerequisites for Milvus integration.

You can use the **Connect Milvus** to connect to the Milvus service inside watsonx.data. In the **Connect your search provider** section, provide the following fields from watsonx.data to enable your assistant to connect to your Milvus instance.

- **GRPC host** and **GRPC port**

  To get the details of **GRPC host** and **GRPC port**, see Connection details.



- **Choose an authentication type**
  - If you select `Basic authentication`, you must provide a **username** and **password**.
  - If you select `watsonx.data API key`, you must provide the corresponding **API key**.
  - if you select `None`, you cannot provide any other authentication details.

  For more information about credentials, see Getting credentials.

Updating your Milvus authentication credentials

Starting with version 25.1.0, IBM watsonx.data uses a new authentication format for Milvus services. This update improves security and aligns authentication across all SaaS environments.

You must update your Milvus username to keep your Milvus connection active and continue retrieving data.

Step 1: Find your new Milvus username

Your new Milvus username uses this format:

`ibmlhapikey_<your watsonx.data username>`

Examples

watsonx.data username	Milvus username
abc@ibm.com	ibmlhapikey_abc@ibm.com
serviceid-abcdef-123456	ibmlhapikey_serviceid-abcdef-123456

If you do not know your watsonx.data username:

1. Go to **Infrastructure manager** in watsonx.data console.
2. Select your Milvus service to open the **Details** page.
3. Click the **Access control** tab.
   Your watsonx.data usernames appear in the first column of the table.

Step 2: Update your Milvus connection in agent builder

Update your Milvus credentials for each assistant that uses a Milvus as a source.

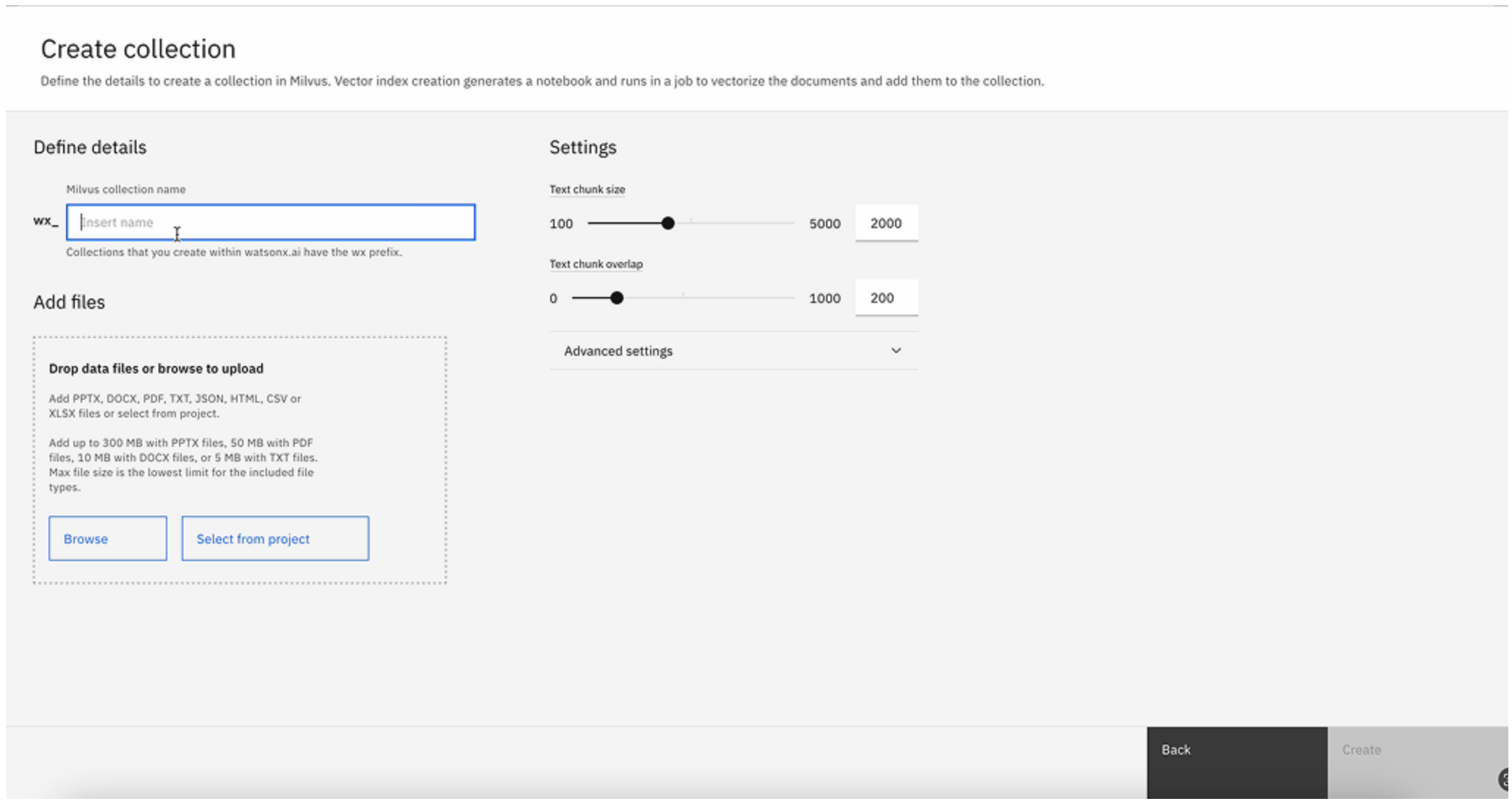
1. Go to **Home** > **Integrations**.
2. Scroll down to the **Extensions** section. In the **Search** tile, click **Add** to open the **Set up a new search integration** window.
3. Select **Milvus**.
4. Record your existing connection information. You need to have a record of your current settings so that you can find them again after you update your Milvus credentials.
5. Go to the **Instance** tab and click **Update details**.
6. Specify your connection details:
   - **GRPC host** and **GRPC port**.
   - **Username:** Use your new Milvus username from Step 1: Find your new Milvus username.
   - **Password:** Use the same password or API key that you used earlier.
     If you are authenticated with a watsonx.data API key, that same key acts as your password.
7. Click **Next**, and reenter your settings.
   - If the connection works, your new credentials are valid.
   - If you see an error, review Step 1: Find your new Milvus username to confirm your username and password.
8. Use your assistant chat window to verify that Milvus search results appear correctly.

Ingesting data into the Milvus vector database through watsonx.ai

After you collect information from watsonx.data, you must ingest data into the Milvus database to use in watsonx Assistant.

In the Milvus window of your assistant, click **Next** to go to **Select data source** and provide the following details:

- In **Database**, select your preferred database.
- In **Choose collection**, select your collection. For more information, see Prerequisites for Milvus integration.



- In **Choose index**, select the index.
- In **Choose embedding_model_id**, select your model.

For more information on the supported embedding models, see Embedding models.

Note: Your selected model must align with the model that you used to create your index.

Configuring the result content

After you connect Milvus by selecting the data source, you can configure how the search response displays in the Milvus window of your assistant. In the **Configure result content** section of **Select data source**, provide the following fields to map the title, body, and URL from Milvus to the search response in the assistant window:

- **Title** Search result title. Use the title, name, or similar type of field from the collection as the search result title. Select something for the title or no search result response is displayed in the Facebook and Slack integrations.
- **Body** Search result description. Use an abstract, summary, or highlight field from the collection as the search result body. Select something for the body or no search result response is displayed in the Facebook and Slack integrations.
- **URL** This field can be populated with any footer content that you want to include at the end of the search result.

Enabling conversational search in Milvus

After you configure the result content, click **Next** to go to **Conversational search** (optional).

Note: Conversational search is available only in the Plus and Enterprise plans of watsonx Assistant.

To activate conversational search, switch the **Conversational Search** toggle to `on`. For more information, see Conversational search.

Defining filter expression

Expand the **Advanced Milvus settings** section to define **Filter**.

You can define the filter as a string to filter the Milvus search results. For more information, see Milvus-filter-search. For more information on filter expression examples, see Filter expression examples.

Tuning your conversational search

You can tune your conversational search's tendency to say "I don't know" and the generated response length.

Use the **Message**, **No results found**, and **Connectivity issue** tabs to customize different messages to share with users based on the successfulness of the

search.

Tab	Scenario	Example message
Message	Search results are returned	I found this information that might be helpful:
No results found	No search results are found	I searched my knowledge base for information that might address your query, but did not find anything useful to share.
Connectivity issue	I was unable to complete the search for some reason	I might have information that could help address your query, but am unable to search my knowledge base at the moment.

Search result messages

You can skip this step if you do not want to change the Milvus instance details. If you want to change the Milvus instance credentials, click the **Instance** tab, change the authentication type, or edit the credentials, and then click **Save**. If you want to change the Milvus instance URL, click the **Update** button next to the URL that takes you to Step 1 to start the initial setup.

Click **Save** and then **Close** to finish the Milvus setup.

Filtering expression examples

The following examples help you to define a filter expression.

Contains

```
$ title like "%action%"
```

This filter expression allows only the search results with title that contains the word `action`.

Doesn't contain

```
$ not (title like "%action%")
```

This filter expression allows only the search results with title not containing the word `action`.

Equal

```
$ "Understand your most and least successful actions"
```

This filter expression allows only the search results with title equal to the phrase `Understand your most and least successful actions`.

Doesn't Equal

```
$ "Understand your most and least successful actions"
```

This filter expression allows only the search results with title not equal to the phrase `Understand your most and least successful actions`.

Nested filter expression

```
$ (title like "%action%") and (url in ["www.url1.com", "www.url2.com"])
```

This filter expression allows only the search results with the title that contains the word "action" and the url in one of ["www.url1.com", "www.url2.com"].

Configuring your assistant to use Milvus

After you configure the Milvus search integration, you must configure your assistant to use Milvus when the response matches no action. For more information about updating **No matches** to use search, see Use search when no action matches.

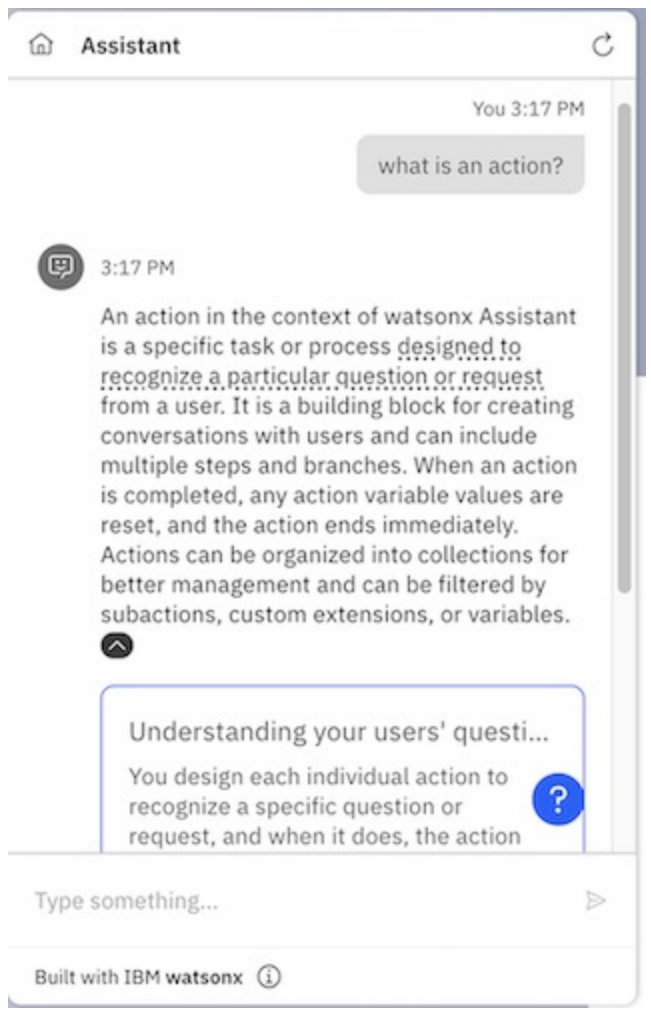
Testing Milvus

You can test search integration with Milvus in actions preview, the preview page, or by using the preview link.

Note: You cannot see proper results in your preview when conversational search is `off`.

In this example, the user asks, `What is an action?`.

A text-based reply from the best results in your knowledge base displays when conversational search is `on`.



Custom service integration setup

Plus Enterprise

A Custom service integration searches for information by using a search capability that you create. You can use a Custom service integration with the conversational search capabilities of your assistant to generate AI responses. This integration supports both server-side and client-side retrieval of information.

> ⚠ **Important:** You can have only one search integration per environment. When you change the existing search integration to other integration types such as IBM Watson® Discovery, Elasticsearch or Milvus, the settings of the existing search integration are overwritten.

Selecting a Custom service

To select a Custom service as the search integration, use one of the following procedures:

- **Selecting Custom service search integration from the Integrations page**

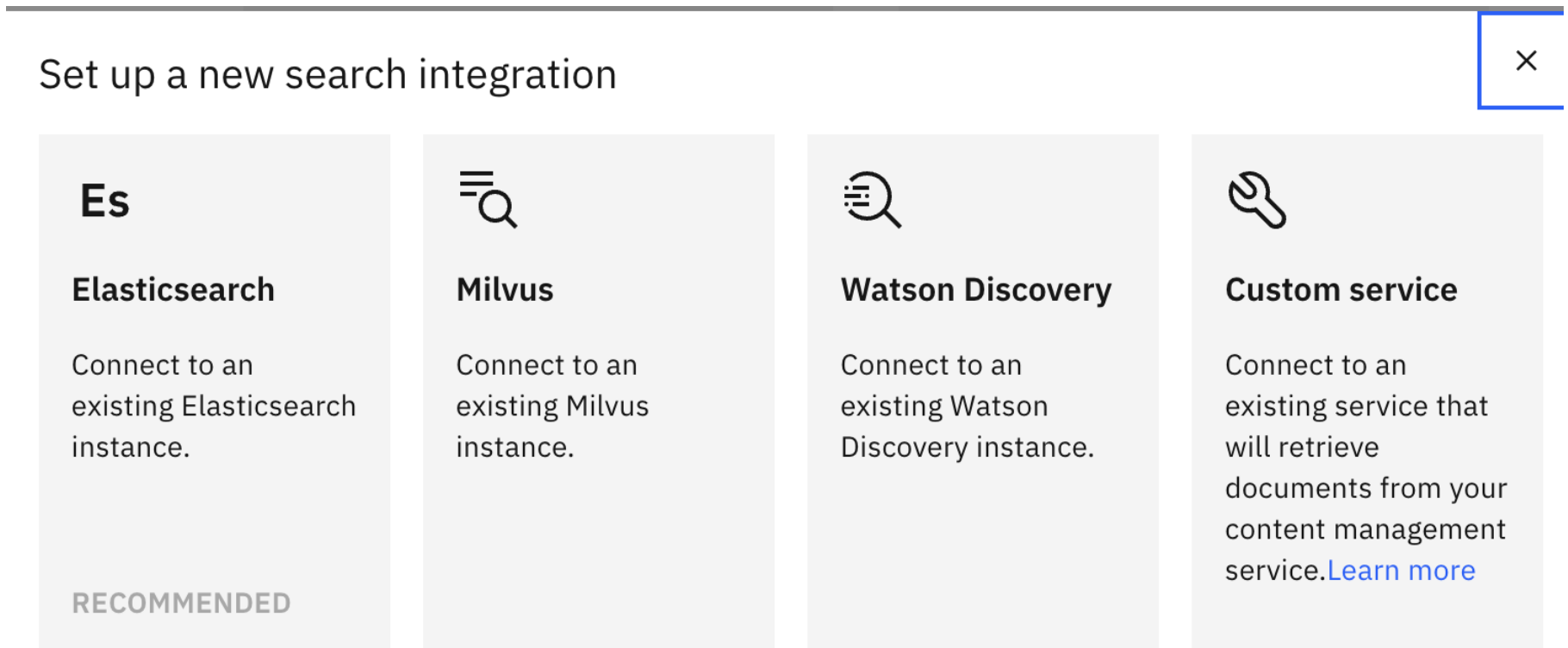
  1. After you create a watsonx Assistant instance, go to **Home** > **Integrations**.
  2. Click **Open** inside the **Search** tile to view the "Open Search" window.
  3. In the "Open Search" window, select the `Draft` option in the dropdown if you want to set up Custom service in your assistant's draft environment. If you want to set up Custom service in your assistant's live environment, select the `Live` option in the dropdown.
  4. In the following "Edit an existing new search integration" window, select the **Custom service** tile.

- **Selecting Custom service search integration from the Environments page**

  1. After you create a watsonx Assistant instance, go to **Home** > **Environments**.
  2. Select the `Draft` tab if you want to set up Custom service in the draft environment. If you want to set up Custom service in the live environment, select the `Live` tab.
  3. In the **Resolution methods** section, click **Add** inside the **Search** tile under **Extensions** if you want to add a Custom service search integration.

     > ☑ **Tip:** If you already added the Custom service search integration, you see the **Open** button instead of **Add** inside the **Search** tile under **Extensions**.

  4. In the `Set up a new search extension` window, select the **Custom service** tile to see the `Search integration` dialog.



Select Custom service

Note: Your assistant has limitations in passing the search parameters directly to your Custom service or retrieving the search results directly from it. Your Custom service search integration must host an accessible web server that implements the search API interface that is provided by the assistant or a client that is configured to call your assistant to provide search results. For more information, see Setting up retrieval systems for a Custom service.

Setting up a Custom service with server credentials

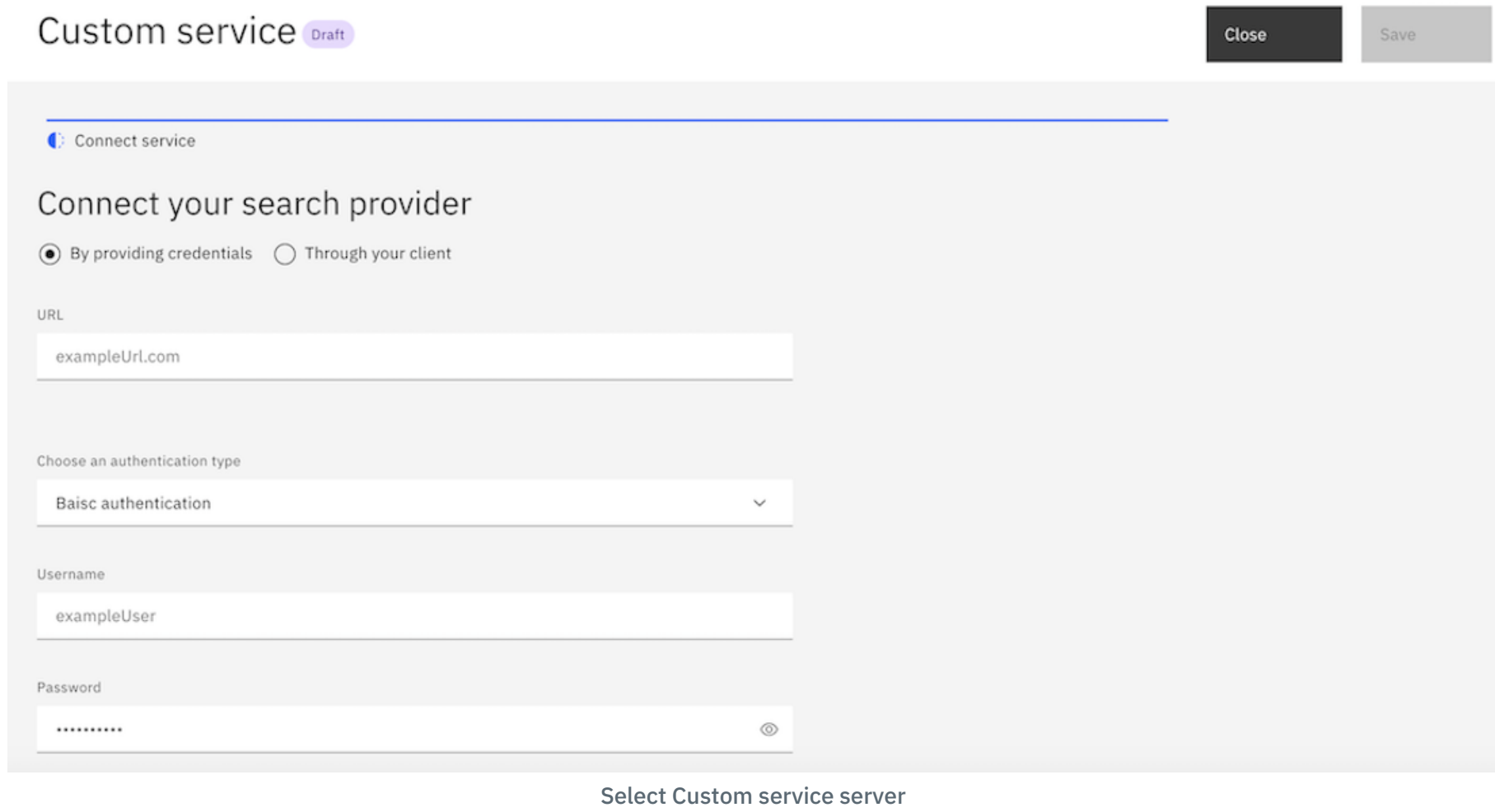
To set up Custom service on your assistant with server credentials, use the following procedure:

1. In the **Connect your search provider** section of the **Custom service** window, select **By providing credentials**. By default, this option is selected.
2. Provide the following fields to enable your assistant to connect to your Custom service instance:
   - **URL**
   - **Choose an authentication type**
     - if you select `Basic authentication`, you must provide a **Username** and a **Password**.
     - if you select `API key`, you must provide an **API key**.
     - if you select `None`, you cannot provide any other authentication details.
3. Click **Next** to go to **Conversational search (optional)**.
4. If you want to activate conversational search, switch the **Conversational Search** toggle to `on`. For more information about conversational search, see conversational search.
5. Filling **Default filter** and **Metadata** is optional. You can place the information in these fields for your server to perform search requests. The metadata must be a JSON object and the default filter can be a text string. You can override the default filter in an action step or dialog node that starts the search. You cannot override the metadata through other options and the metadata you provide applies to all uses of this integration. For more information, see filling default filter and metadata for server .
6. Use the **No results found** and **Connectivity issue** tabs to customize different messages to share with users based on the success of the search.

Tab	Scenario	Example message
No results found	No search results are found	`I searched my knowledge base for information that might address your query, but did not find anything useful to share.`
Connectivity issue	I was unable to complete the search for some reason	`I might have information that could help address your query, but am unable to search my knowledge base at the moment.`

Custom service search result messages

7. Click **Save** and then **Close** to end the custom service set up with server credentials.



Select Custom service server

Setting up a Custom service through your client

To set up Custom service on your assistant through your client, use the following procedure:

1. In the **Connect your search provider** section of the **Custom service** window, select “Through your client”.
2. Click **Next** to go to **Conversational search (optional)**.
3. If you want to activate conversational search, switch the **Conversational Search** toggle to `on`. For more information about conversational search, see conversational search.
4. Filling **Default filter** and **Metadata** is optional. You can place the information in these fields for your server to perform search requests. The metadata must be a JSON object and the default filter can be a text string. You can override the default filter in an action step or dialog node that starts the search. You cannot override the metadata through other options and the metadata you provide applies to all uses of this integration. For more information, see filling default filter and metadata for client .
5. Use the **No results found** and **Connectivity issue** tabs to customize different messages to share with users based on the success of the search.

Tab	Scenario	Example message
No results found	No search results are found	I searched my knowledge base for information that might address your query, but did not find anything useful to share.
Connectivity issue	I was unable to complete the search for some reason	I might have information that could help address your query, but am unable to search my knowledge base at the moment.

Custom service search result messages

6. Click **Save** and then **Close** to end the Custom service set up in the client-side.

Setting up Milvus for Custom service

Milvus is a vector database that you can use for handling large-scale datasets. For applications that require real-time search capabilities and numerous concurrent users, you can use Milvus, which has a distributed architecture, high performance, and flexible data model.

You can directly integrate with your watsonx.data Milvus for Conversational Search rather than using a custom service. For more information, see Milvus search integration setup.

You need to use the Custom service search, to avail more advanced search capabilities with Milvus, such as:

- Flexibility to use any Milvus that is supported and watsonx.ai supported embedding models. For a list of embedding models supported by Milvus, see Milvus supported embedding models.
- Multi-vector hybrid search.
- Reranking.

Setting up Milvus with server credentials

1. For setting up watsonx.data Milvus, see the Guide for setting up the search integration with watsonx.data Milvus.
2. For general setup on your assistant with server credentials, see Setting up a Custom service with server credentials.
3. For more information on examples and references for Milvus, see Example with Milvus.

Setting up Milvus through your client

Set up Milvus on your assistant through your client by following the steps in Setting up a Custom service through your client.

Setting up retrieval systems for a Custom service

To use a Custom service with your search integration, you must integrate your search capability by providing a server or by having the client that calls your assistant to provide search results. You can use your own retrieval if the retrieval schema matches with the schema that is provided by your assistant. If your retrieval schema does not match with the assistant's schema, you must provide a wrapper that does the schema mapping. You can deploy the wrapper as a service or your chat client can start it. Building a wrapper is useful when you want to combine a different source or invoking libraries or services that do not comply with the schema for assistant search results.

Setting up a server for Custom service retrieval

A server for Custom service retrieval must implement the following API:

Query: `POST <server_url>`

Request

```
{
   "query": "<QUERY>",
   "filter": "<FILTER>", // optional
   "metadata": {
      // optional, you can fill any information here
   }
}
```

Response

```
{

  "search_results": [
    {
      "result_metadata": { // optional
        "score": <SCORE as a number>
      },
      "title": "<TITLE>",
      "body": "<BODY>",
      "url": "<URL>", // optional
      "highlight": { // optional, will be used instead of "body" for Conversational Search if provided
        "body": [
          "<HIGHLIGHT1>",
          "<HIGHLIGHT2>",
           ...
        ]
      }
    }
  ]

}
```

⚠ **Important:** The metadata in the request and the entire response object must not exceed 100 KB.

Setting up a client for Custom service retrieval

You can see the following API response from the /message API requesting search at run time:

```
{
   "output": {
      "intents": [ ... ],
```

```
      "actions": [
        {
          "type": "search",
          "query": "<QUERY>",
          "filter": "<FILTER>",
          "metadata": { // optional
            /* you can use any JSON object here */
          }
        }
      ]
    }
}
```

Whenever the chat client receives a response with that form (it has an entry in the `output.actions` list of type `search`), it passes the results back to the assistant through the next call to the `/message API` as follows:

```
{
   "input": {
      "message_type": "search_results",
       "search_results": [
        {
          "result_metadata": { // optional
            "score": <SCORE as a number>
        },
        "title": "<TITLE>",
        "body": "<BODY>",
        "url": "<URL>", // optional
        "highlight": { // optional, will be used instead of "body" for Conversational Search if provided
          "body": [
          "<HIGHLIGHT1>",
          "<HIGHLIGHT2>",
          ...
          ]
        }
      }
   ]
  }
}
```

⚠ **Important:** Your assistant response limit cannot exceed 100 KB. If your assistant gets a `search_results` message with a body that exceeds 100 KB, it returns a 400 response.

Processing the search results in conversational search

When you set up your Custom service by providing server credentials or by sending results from your client and if you enable conversational search with your Custom service, you get the following behavior:

1. Conversational search iterates through the search results from first to last.
2. From each search result:
   - If you don't have a `highlight.body` list, it takes the `body` as a text snippet.
   - If a `highlight.body` list is present, it takes each element in that list as a text snippet.
3. After discarding the duplicate text snippets, it continues to iterate through the search results and `highlight.body` lists until it has 5 text snippets.
4. Conversational search applies a pre-generation filter model to compare the query and the search results to judge the relevance of the results to the query. If pre-generation filter model produces two scores:
   - A low score, conversational search returns an *I don't know* signal. For more information on *I don't know* signal, see conversational search.
   - A high score, conversational search sends the snippets along with the corresponding titles to the generative AI model to generate an answer.
5. If the text is too long for the generative AI model to process, it repeatedly discards the last snippet or title pair until it is short. When it has no text, the search fails.
6. Conversational search applies the response to the post-filter model. If the post-filter model produces two scores:
   - A low score, conversational search returns an *I don't know* signal.
   - A high score, conversational search returns the generated response along with all the `search_results` to the calling application.

Your assistant follows the same process for the other search options like Elasticsearch and IBM Watson® Discovery.

Adding separate fields for `body` and `highlight.body`

When the `highlight.body` is present, it is used to generate conversational search answers, else the `body` is used. Both are passed to the client as part of the search results object and the client provides context for the answer. For example, the built-in web chat shows the `body` text. If you click a citation card for some search result, you can't see a URL for that search result.

Recommendations for using the fields:

- If the search technology returns short portions of documents, use those portions of the documents in the `body` field and omit the `highlight.body`. For example, many vector database solutions store only 512 token segments of documents.
- If the search technology returns both short portions of documents ("passages" or "highlights" or "snippets") and the full text of the documents, use the short portions in the `highlight.body` field and use the full text in the `body` field.

Adding optional user-defined metadata

For both clients and servers, the schemas for results include a `metadata` field and a `result_metadata` field.

Recommendations for using the fields:

- The `metadata` field sends the configuration information to the search capability. Some examples of configuration information that is used for a search capability include index names, embedding model names, requested passage length, fields to boost, and so on. The following reasons explain why it is helpful to use the `metadata` field to pass configuration information to the server or client:
  - Multiple assistants can use the same server or client code but with a different configuration.
  - Even with one assistant, it is easy to update configurations through the assistant interface.
- The `result_metadata` field sends additional information about a search result from the server or client to the assistant. The assistant passes the information as part of the `search_results` object in the final response. Calling applications use the additional information. For example, when the `result_metadata` sends the URLs for images in the search result, the calling application renders the images along with the response.

Coveo search extension setup

You can use Coveo to create and manage source documents. The Coveo Search API can issue search queries on an index. It is a configurable search that you can customize based on your use case. You can access Coveo search through an extension to your assistant.

To set up the extension for Coveo search:

Add an API key

In the Coveo Administration Console, add an API key. For detailed instructions, see Add an API Key in the Coveo documentation. Ensure that the API key is set to enable search and to allow `Execute queries`.

Download the OpenAPI specification

Download the OpenAPI specification file: coveo.openapi.json. Use this file to add the extension to your assistant.

The OpenAPI specification defines the following method:

- `GET /rest/search/v2`: Search for content in a set of sources or documents.

For more information about the endpoints, see Perform a Query in the Coveo documentation.

Create and add the extension

1. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
2. After you build the Coveo extension, and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Coveo API key to authenticate. For general instructions on adding any custom extension, see Adding an extension to your assistant.

Add the Coveo search starter kit action template

1. Open the **Actions** page.
2. If you have no actions, select **Create a new action**. If you already have some actions, select **New action**.
3. On **Create an action**, select **Quick start with templates**.

> **Note: Quick start with templates** is available in English-language assistants only.

4. On **Quick start with templates**, add the Coveo search starter kit.

Edit system actions

1. Click **Set by assistant** and open the **No matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to a subaction**, and select the **Coveo search** action.
4. If you are not connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No matches**.

Using your Coveo search extension

Issue a query to your assistant. If no action matches that query, then it uses Coveo to produce search results.

Google custom search extension setup

You can access Google search through an extension to your assistant that uses the Google Programmable Search Engine. It is a configurable search that you can customize based on your use case.

To set up the extension for Google search:

Get Search Engine ID and API key

Create a Google Programmable Search Engine. Then, get its Search Engine ID and an API key. For detailed instructions, see Create Programmable Search Engine in the Google Programmable Search Engine documentation.

Download the OpenAPI specification

Download the OpenAPI specification file: google-custom-search-openapi.json. You use this file to add the extension to your assistant.

The OpenAPI specification defines the following methods:

- `GET /customsearch/v1`: Search for content over the entire web.
- `GET /customsearch/v1/siterestrict`: Search for content over a specific collection of websites.

For more information about the endpoints, see Custom Search or Custom Search Site Restricted.

The endpoints have the same arguments and responses, but with differences:

- *Custom Search Site Restricted* is restricted to searching 10 or fewer websites, each of which can have an unlimited number of pages.
- *Custom Search* can support any number of websites that is indexed by Google, but has a daily query limit.

For a typical assistant focused on a specific topic, it is usually only necessary to search a single website or a few websites. *Custom Search Site Restricted* is a better fit since it doesn't have a limit on the number of queries that can be run per day. Assistants that need to search more than 10 websites need to use *Custom Search* instead.

Create and add extension

1. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
2. After you build the Google custom search extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your Google programmable search engine API key to authenticate. For general instructions on adding any custom extension, see Adding an extension to your assistant.

Add the Google custom search starter kit action template

1. Open the **Actions** page.
2. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
3. On **Create an action**, choose **Quick start with templates**.

> **Note: Quick start with templates** is available in English-language assistants only.

4. On **Quick start with templates**, add the Google custom search starter kit.

Edit system actions

1. Click **Set by assistant** and open the **No matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to a subaction** and choose the **Google search** action.
4. If you aren't connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No matches**.

Using your Google custom search extension

Issue a query to your assistant. If no action that matches that query, then it uses Google to produce search results.

Limit on search results size

watsonx Assistant has a 100 kb limit on the size of information that is stored in context variables, which includes search results. If the results from your extension exceed that limit, the action can fail without any visible warning or error. Typically a long delay occurs and then there is no response. This failure rarely happens with the Google custom search extension, but it might happen if you are searching a site with large volumes of metadata that is returned by Google custom search. If you think that this might be a problem, try running the query in an API testing tool like curl, Insomnia, or Postman. Check how many bytes of data you are getting as search results. If the total is at or near 100 kb, you might be able to work around the issue by reducing `num_of_results` and getting fewer results for each query or by excluding sites or pages with large volumes of metadata.

For more information, see Limit on Size of Search Results in a starter kit for IBM Watson® Discovery.

NeuralSeek extension setup

NeuralSeek by Cerebral Blue is a combined search and natural-language generation system that is designed to make conversational AI feel more conversational. It requires that you load all your content into IBM Watson® Discovery. Then, when a user asks a question, it has Discovery search for multiple relevant documents and then it generates a natural-language answer that uses the contents of those documents. In some cases, the answer might be taken directly from a single document, and in others, the answer can include information from multiple sources that are combined into a single coherent statement. For each query, NeuralSeek returns a single answer and a confidence score. In most cases, it also returns a URL of a document that influenced the answer, which might be one of several documents.

To set up the extension for NeuralSeek search:

Set up IBM Watson® Discovery

1. You need an instance of IBM Watson® Discovery. Because NeuralSeek can modify your data as needed to make it more effective, make sure it is not an instance with important data that you are using for other purposes.
2. In Discovery, create a project and load the documents that you want to use.

Get the NeuralSeek OpenAPI specification and API Key

1. You also need an instance of NeuralSeek on IBM Cloud.
2. In NeuralSeek, open the **Configure** page and enter your information in the **Discovery instance details** section.
3. On the **Integrate** page, and click the **OpenAPI file** link to download the `NeuralSeek.json` OpenAPI specification file configured for your instance.
4. Also on the **Integrate** page, copy the API key for NeuralSeek. The API key for NeuralSeek is not the same as the API key for Discovery that you used on the **Configure** page. The API key for NeuralSeek is only available on the **Integrate** page.

Note: Simple instructions for setting up NeuralSeek are available on the **Integrate** page. You can also follow those instructions to get NeuralSeek working with watsonx Assistant.

Create and add extension

1. In your assistant, on the **Integrations** page, click **Build custom extension** and use the OpenAPI specification file to build a custom extension. For general instructions on building any custom extension, see Building a custom extension.
2. After you build the NeuralSeek extension and it appears on your **Integrations** page, click **Add** to add it to your assistant. Use your NeuralSeek API key to authenticate. For general instructions on adding any custom extension, see Adding an extension to your assistant.

Add the NeuralSeek starter kit action template

1. Open the **Actions** page.

2. If you have no actions, choose **Create a new action**. If you already have some actions, choose **New action**.
3. On **Create an action**, choose **Quick start with templates**.

> **Note:** **Quick start with templates** is available in English-language assistants only.

4. On **Quick start with templates**, add the NeuralSeek search starter kit.

Edit system actions

1. Click **Set by assistant** and open the **No matches** action.
2. Delete the two default steps.
3. Add a step. Set **And then** to **Go to a subaction** and choose the **NeuralSeek search** action.
4. If you aren't connecting your customers to a live agent, you might want to edit the **Fallback** action in the same way as **No matches**.

Using your NeuralSeek extension

Issue a query to your assistant. If no action that matches that query, then it uses NeuralSeek to produce search results.

For many use cases, NeuralSeek alone is enough to deploy an assistant. If you are happy with your assistant, you might want to deploy it for real-world use right away. Use the **Analyze** page in watsonx Assistant or the **Curate** page in NeuralSeek to see what kinds of questions users are asking and actions for the common user requests. The **Curate** page can automate the creation of new actions and generate new example utterances that trigger those actions. It can also merge any existing actions with the actions that it generates so you can update an existing assistant. For more information, see the NeuralSeek documentation

Search integration enhancement

Use the following enhancements for the search integration to enhance the assistant responses to customer queries.

- Search trigger
- Use search when no action matches
- Use Conversational search

Search trigger

The search integration is triggered from an action step. By default, the action sends the most recently submitted user message as the search query. However, you can use the search settings within the action step to change the custom search query and custom results filter, which helps you to get accurate results.

For example, the conversational flow might collect information about the type of device a customer wants to buy. When you know the device model, you can then send a model keyword in the query that is submitted to the search integration to get better results.

To configure the search query, complete the following steps:

1. In the *And then* field of the step where you want the search to be triggered, choose **Search for the answer**.
2. Click **Edit settings**.
3. Add values to one or both of the following fields:
    - **Custom search query**. Add a word or phrase that you want to submit to search integration as the query string for the search.

      For example, you can specify a string such as, `What cities do you fly to?`.

      For a more dynamic string, you can include a variable. For example, `Do you have flights to ${destination}?`

      You are effectively defining the value that is used by the search integration API as the `natural_language_query` parameter. For more information about defining query values for IBM Watson® Discovery, see Query parameters Discovery. For more information about defining query values for Elasticsearch, see Query parameters Elasticsearch.

      If you don't specify a text string, the action sends the most recently submitted user message as the search string.

      If you want to use the original customer message that triggered the action as the query string instead, you need to plan ahead. You can follow these steps:

      1. Create a session variable to store the initial user input. For example, named `original message`.

2. In Step 1, meaning the first step after the action trigger, set the value of the session variable. For more information about session variables, see Creating a session variable.
3. Set the value of the variable by using an expression that looks like this: `<? input.text ?>`.

This expression captures the complete message that was submitted by the customer. As a result, your variable captures the customer message that triggered this action.

1. Add the session variable to the *Custom query* field (for example, `${original_message}`).

- **Custom results filter**: Add a text string that defines information that must be present in any of the search results that are returned.

  You are effectively defining the value that is used by the search integration API as the `filter` parameter. For more information about defining filter values for IBM Watson® Discovery, see Discovery filter. For more information about defining filter values in Elasticsearch, see Elasticsearch filter

  The syntax to use for the filter value is not intuitive. Here are a few examples of common use cases:

  - To indicate that you want to return only documents with positive sentiment, for example, specify `enriched_text.sentiment.document.label:positive`.
  - To filter results to include only documents that mention `Boston, MA`, specify `enriched_text.entities.text:"Boston, MA"`.

    If you add both a query and a filter value, the filter parameter is applied first to filter the data collection documents and cache the results. The query parameter then ranks the cached results.

    a. If you want the search for an answer to be the last step in the action, select **End the action after returning results**.

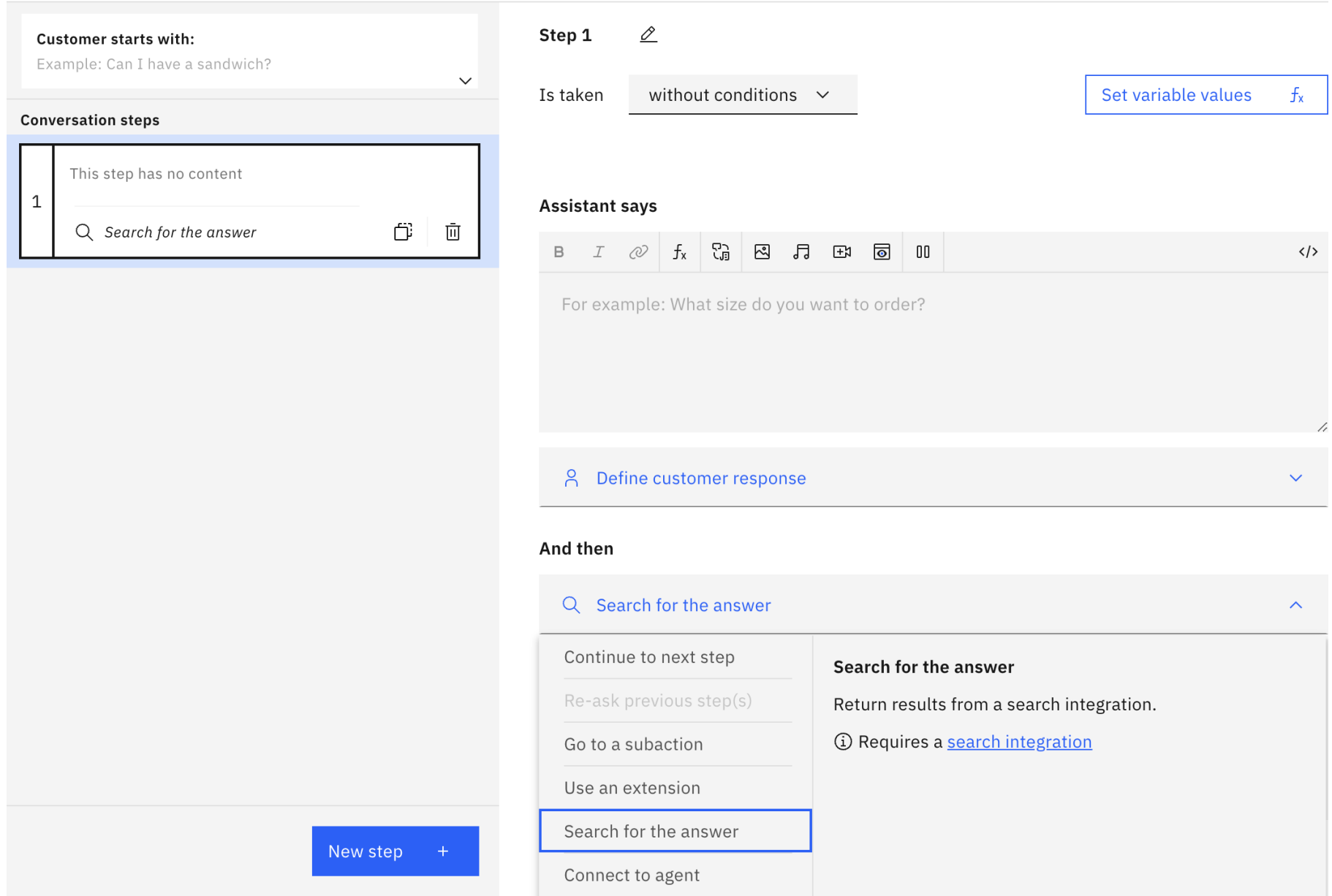
4. Click **Apply**.

Use search when no action matches

You can use the search integration with the built-in No matches capability. By adding search to **No matches**, you can have your assistant refer to search when a customer asks a question that isn't addressed by an existing action.

To update *No matches* to use search:

1. In your assistant, click **Actions**, then click **Set by assistant**.
2. Click **No matches** to open it in the editor.
3. Click **New step**.
4. In the **And then** section, click **Continue to next step**, then choose **Search for the answer**.



Configure no matches to use search

5. Close **No matches**. Your assistant uses search to provide customers with potentially useful answers, if the customer question does not trigger any of the existing actions.

Use Conversational search

Conversational search uses the large language models (LLMs) to recognize and respond to customer queries. You can enable this feature in the search integration to improve the assistant responses by using simple conversations.

For more information about Conversational Search, click Conversational search.

Managing and organizing your actions

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.

Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Creating skill-based actions

Skill-based actions, also known as conversational skill actions, allow assistants to connect and start tasks from third party applications and services. The assistant then can report the status of the task and interact with the user to complete the given task. For more information about skills, see Overview of apps and skills in the watsonx Orchestrate documentation.

Skill-based actions require a provider that connects to watsonx Assistant and provides access to the external application's or service's features. In watsonx Orchestrate, you can find Pre-built skills and apps that allow you to access the functionalities of many services, but you can't register your own application to use it as a skill provider.

In watsonx Assistant, you can use your own application as a skill-based action provider and use them to enhance the user's experience with your assistant, providing access to functions that it would not be able to do otherwise.

Before you begin

To create your own skill-based actions you must first complete the following requirements:

1. Implement an API that complies with the API specifications for skill-based action providers.
2. Register a skill-based action provider in watsonx Assistant.

Implementing the API for the skill-based action provider

For more information about the API specifications for skill providers, see Conversational Skills: Masterdoc for API Endpoints to Implement by the Pro-Code Conversational Skill Client.

You can also use Java SDK for conversational skills. For more information, see Conversational Skills: Java-based conversational skill for watsonx Assistant.

Registering a skill-based action provider

After you create the endpoints, you must register your skill-based action provider in watsonx Assistant. For more information about how to accomplish that, see Register a Conversational Skill Provider.

If you need more examples of authentication in different schemas, see Create a conversational skill provider.

Creating skill-based actions

There are two methods to create a skill-based action in watsonx Assistant. You can create them by using the API or by using the watsonx Assistant user interface. Choose the option that is most suitable to your needs.

Creating skill-based actions by using the API

To create a skill-based action by using the API, you must first create an OpenAPI specification of the skill that references the endpoints of your skill provider.

You can use any OAS editor or the **watsonx Orchestrate OpenAPI Builder** to complete this task. For more information, see Building OpenAPI specifications for skills in the watsonx Orchestrate documentation.

After that, you can import the skill by using the Import skills endpoint in the watsonx Assistant API. When that's done, you can then use the skill in your assistant.

Creating skills by using the UI

You can also use the watsonx Assistant UI to create the skill-based action.

> ⚠ **Important:** To create a skill based action from the UI, your skill provider's API must implement the list conversational skills endpoint.

> **Note:** Actions from skills don't support getting input values from other actions or mapping output values to other actions.

To create a skill-based action, complete the following steps:

1. In watsonx Assistant, go to **Actions** > **New action** > **Skill-based action**.
2. In the **Build an action from a skill** page, click the skill to which you want to link the action to, making the skill the foundation to your action.

   > **Note:** You must connect to the app of the skill for it to appear as an option in this step. For more information, see Managing connections and connecting to apps.

3. In the **Name your action** field, enter a name for your action.
4. In the **Add action conditions**, you can add or edit conditions for the action trigger.

   > **Tip:** Adding conditions to an action

5. In the **Enter a phrase** field, enter the phrase that customers must type or ask to trigger the skill. For example, `I want to pay my electricity bills`.
6. You use the following options on the upper-right corner of the page to improve the assistant experience:
   - **Action response mode**

     Defines the response mode for your assistant such as **Clarifying** or **Confident**.
   - **Action notes**

     Opens the **Action notes** window to add a description, documentation, comments, or any other annotations to help you track your work as you build an action.
   - **Action settings**

     Opens the **Action settings** window to enable or disable **Ask clarifying question** and **Change conversation topic** configurations in action.
7. Click the **Save** button to save the action.

Your assistant can now use the newly created action.

Using skill-based actions

To test and use your skill-based actions, you can start a new conversation with your assistant and follow the conditions that you defined when you created the action. Once you meet the conditions, enter the phrase that triggers the action.

The assistant then attempts to turn that action into a conversation, where you can check the status of the task and interact with the assistant to complete the action successfully.

Duplicating an action

You can duplicate an action to reuse information in a new action. When you duplicate an action, the new action includes everything except example phrases.

To duplicate an action, click the overflow menu on the action you want and select **Duplicate**.

Duplicate action

Copying an action to another assistant

You can copy an action from one assistant to another when you want to reuse the action content and configuration in the destination assistant.

When you copy the action to the destination assistant, references to actions, saved responses, and variables in the source assistant are copied.

To copy an action from one assistant to another assistant:

1. Click the **Overflow** menu ⋮ on the action that you want and select **Copy to an assistant**.
2. Click the **drop-down** icon to select the destination assistant for the copy.
3. In the **Review references** page, review the list of references such as action, variables, and saved responses and their status.

Type of references	Conflict status	Outcome
◦ Actions ◦ Variables ◦ Saved responses	No	By default, it saves a new reference to the target assistant.
◦ Actions ◦ Variables ◦ Saved responses	Yes	You can select one of the following options: ◦ Overwrite: Overwrites the existing action in the destination assistant. ◦ Skip: Skips copying the action to the destination assistant.

Conflict status for references

Tip: If you want to overwrite all the conflicting items in the destination assistant, click the **Overwrite all** button. If you want to skip copying all the conflicting items to the destination assistant, click the **Skip all** button.

4. Click **Apply** to finish copying the action to the other assistant.

Uploading or downloading all actions

You can upload or download all your actions as a JSON file.

Downloading

To back up all your actions, download a JSON file and store it.

1. On the **Actions** page, click **Global settings** ⚙.
2. On the **Upload/Download** tab, click the **Download** button.

Uploading

To reinstate a backup copy of actions that you exported from another service instance or environment, import the JSON file of the actions you exported.

Important: If the watsonx Assistant service changes between the time you export the actions and import it, due to functional updates that are

regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before.

1. On the **Actions** page, click **Global settings** ⚙.
2. On the **Upload/Download** tab, drag and drop a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

⚠ **Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

Uploading intents as actions

If you created intents in the classic experience, you can migrate your intents to actions. When you upload intents, each intent is created as a new action. All phrases corresponding to an intent are created as example phrases for the new action. This can provide a helpful starting point when you are ready to start building actions.

1. Download the intents that you want to migrate to actions from the classic experience. For more information, see Downloading intents. The format for each line in the file is as follows:

   ```
   $ <phrase>,<intent>
   ```

   where `<phrase>` is the text of a user example phrase, and `<intent>` is the name of the intent. For example:

   ```
   $ Tell me the current weather conditions.,weather_conditions
   Is it raining?,weather_conditions
   What's the temperature?,weather_conditions
   Where is your nearest location?,find_location
   Do you have a store in Raleigh?,find_location
   ```

2. From the main actions page, click the **Upload** icon ⤒.
3. Select the intents file that you downloaded.

   The file is validated and uploaded, and the system trains itself on the new data.

   The intents in column 2 are created as new actions, and the phrases in column 1 are created as example phrases for the corresponding action. For example, if you upload the example from step 1, two new actions are created for the `weather_conditions` and `find_location` intents. The underscores (`_`) in the intent names are replaced with spaces, for example, the `weather_conditions` intent becomes the `weather conditions` action.

   In this example, the `weather_conditions` action will have three example phrases: `Tell me the current weather conditions.`, `Is it raining?`, and `What's the temperature?`. The `find_location` action will have two example phrases: `Where is your nearest location?` and `Do you have a store in Raleigh?`.

Filtering actions

IBM Cloud

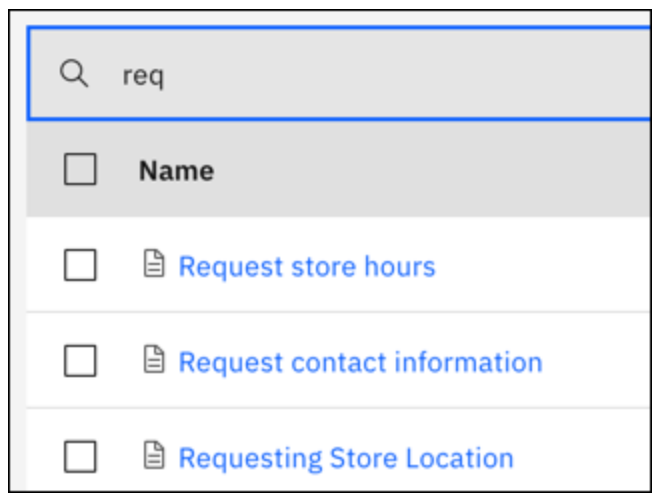
You can locate specific actions by filtering the list.

You can filter by:

Filter	Description
Name	All or part of the action name
Subaction	You can filter by subaction by: • Choosing a subaction from the dropdown list. The list shows other actions that trigger, or call, that subaction in *Go to a subaction* at the end of a step. • Clicking the **Show all subactions** checkbox. The list shows actions that are triggered, or called, by other actions in *Go to a subaction* at the end of a step.
Extension	Choose a custom extension to see all the actions that use it. Or, choose **Any** to see all the actions that use at least one custom extension.
Variable	Choose a variable to see all the actions that use it.

Filters

To filter by all or part of a name, make an entry in the **Filter by name** field. The list is filtered as you type.



To filter by action, extension, or variable:

1. Click the **Filters** icon to see the filter fields.
2. Use the **Subaction**, **Extension**, or **Variable** dropdown lists, and then click **Apply**.
3. Or, click the **Show all subactions** checkbox, and then click **Apply**.

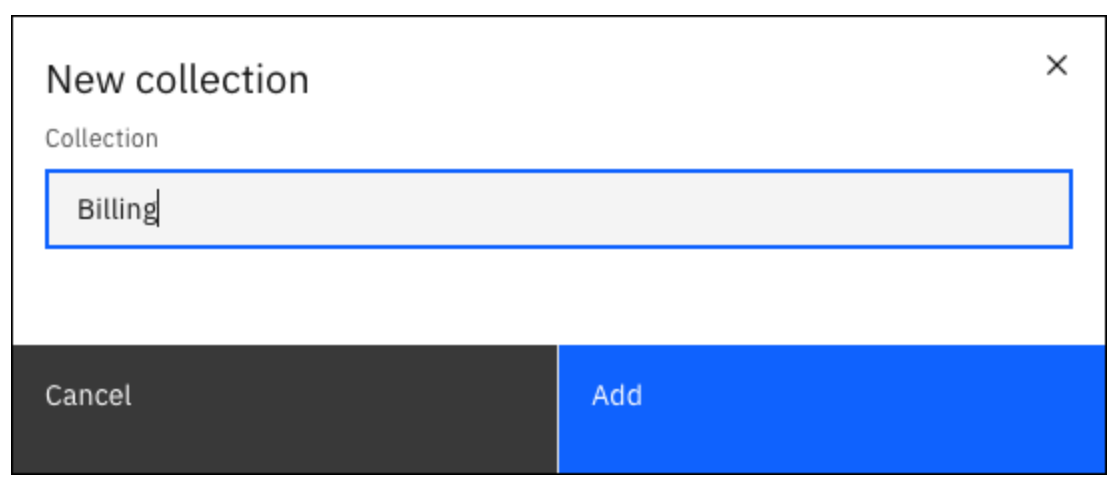
Organizing actions in collections

You can use a *collection* to organize your actions. You can put actions into folder-style groups based on whatever categorization you need at your organization, such as by use case, internal team, or status.

Adding a collection

To add a collection:

1. From the **Actions** page of your assistant, click **New collection**.
2. Enter a collection name of up to 64 characters. Each collection name must be unique.



New collection

3. Click **Add**. The new collection is listed on the **Actions** page.

Collection

You can also add a collection and move actions into it at the same time:

1. Select one or more actions by clicking the checkbox next to each action that you want to include.
2. Click **Add to new collection**.
3. Enter a collection name of up to 64 characters. Names don't need to be unique.

4. Click **Add**. The new collection is listed on the **Actions** page. Open the collection to see the actions you selected.

Moving actions into a collection

You can move one or more actions into a collection. You can also move actions from one collection to another.

To move an action into an existing collection:

1. Select one or more actions by clicking the checkbox next to each action that you want to include.
2. Click **Move to**.
3. Select a collection, then click **Move**.

To move actions from one collection to another:

1. Open a collection.
2. Select one or more actions by clicking the checkbox next to each action that you want to include.
3. Click **Move to**.
4. Select a collection, then click **Move**.

Removing actions from a collection

You can remove an action from a collection:

1. Open a collection.
2. Select one or more actions by clicking the checkbox next to each action that you want to include.
3. Click **Remove from collection**.
4. Open the **Created by you** list to see the actions you removed.

Renaming a collection

To rename a collection:

1. From the **Actions** page of your assistant, click the overflow menu icon ⋮.
2. Click **Rename**.
3. Enter a new name of up to 64 characters. Names don't need to be unique.
4. Click **Rename**. The renamed collection is listed on the **Actions** page.

Deleting a collection

You can delete a collection. No actions in the collection are deleted. They are moved into the **Created by you** list.

To delete a collection:

1. From the **Actions** page of your assistant, click the overflow menu icon ⋮.
2. Click **Delete**.
3. In the confirmation message, click **Delete**.

Calling a custom extension

An extension is an integration with an external service. By calling an extension from an action, your assistant can send requests to the external service and receive response data it can use in the conversation.

For example, you might use an extension to interact with a ticketing or customer relationship management (CRM) system, or to retrieve real-time data such as mortgage rates or weather conditions. Response data from the extension is then available as action variables, which your assistant can use in the conversation.

For information about how to build a custom extension, see Build a custom extension.

Calling the extension from a step

To call a custom extension from an action:

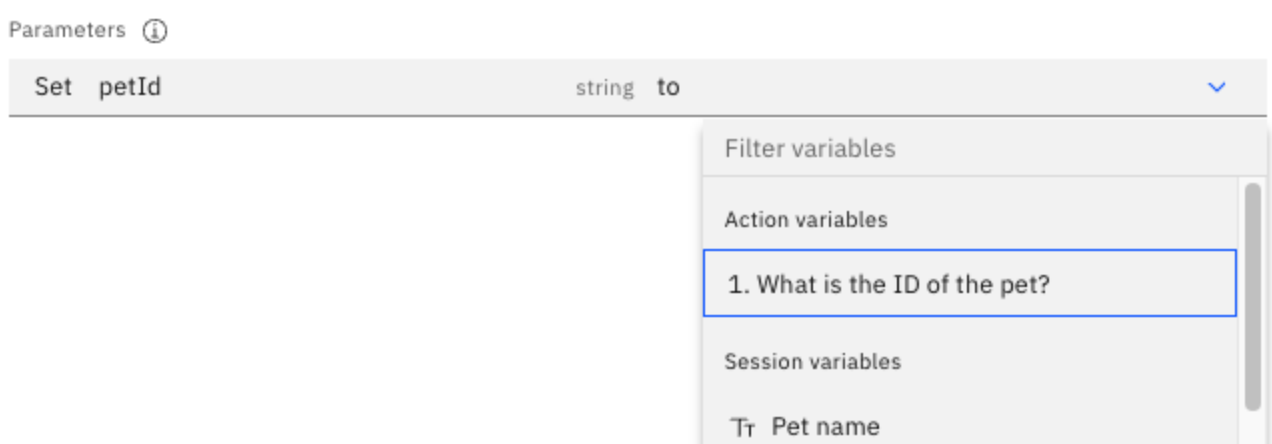
1. In the action editor, create or open the step from which you want to call the extension.
2. **Optional:** In the **Assistant says** field, type a message to be shown to the customer before the extension is called (for example, `Please wait while I retrieve your account balance...`).

   The output from this step is sent to the channel with the global context variable `skip_user_input` set to `true`. This variable tells the channel to display the message but *not* to prompt the customer for a reply. Instead, the channel sends an empty message, enabling the assistant to proceed with the call to the extension.

   > **Note:** All built-in channel integrations (such as the web chat) respect the `skip_user_input` context variable. If you are using the API to develop a custom client, it is your responsibility to include logic checking for this variable. For more information, see Processing user input.

3. In the step editor, click **And then**.
4. Click **Use an extension**.
5. In the **Extension setup** window, specify the following information:
   - In the **Extension** field, select the extension you want to call.
   - In the **Operation** field, select the operation you want to perform. (An *operation* is a method or function supported by the extension.)
6. Specify values for each of the required input parameters. A *parameter* is an input value sent to an operation, such as the ID of a customer record you want to retrieve or the location to use for a weather forecast.

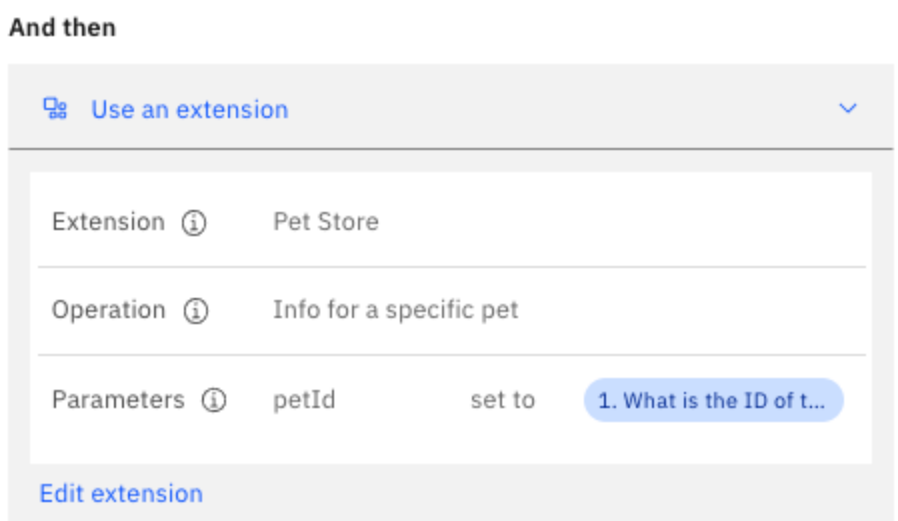
   To assign a value to a parameter, click the input field for the value. You can then select from the list of available variables or write an expression to specify the value.



   Each parameter has a data type (such as *number* or *string*). The variable you select must be compatible with the data type of the parameter; for more information, see Compatible variables for parameters.

   You must specify values for all required parameters before you can proceed.
7. If you want to specify a value for any optional parameters, click **Optional parameters**. You can then repeat this process for each optional parameter you want to use.
8. Click **Apply**. (If the **Apply** button is not available, check to make sure you have specified values for all required parameters.)

The **And then** section of the step editor now shows an overview of the call to the extension:



If you need to make changes, click **Edit extension** to reopen the **Extension setup** window.

Compatible variables for parameters

To pass an input parameter value for an operation, you must select a compatible action variable or session variable.

An action variable contains a value that is based on a customer response in a previous step. A session variable might have a value based on a customer response or a value defined by an expression. (For more information about action variables and session variables, see Using variables to manage conversation information.)

When you assign a value to a parameter, the variable you choose must be compatible with the data type of the parameter. (For example, a *number* parameter must be assigned a numeric value rather than text.)

The following table shows the possible customer response types and the parameter data type compatible with each.

Customer response type	Compatible data types	Notes
options	`string`	A selected option is always treated as a string, even if it is a numeric value.
number	`number` `integer`	A floating-point number passed as the value for an `integer` parameter might cause an error, depending on the behavior of the REST API.
date	`string`	Dates are rendered as `YYYY-MM-DD`.
time	`string`	Times are rendered as `HH:MM:SS` in 24-hour format, converted to the user's time zone.
Currency	`number` `integer`	
Percent	`number` `integer`	A percent value is passed as an integer (so `75%` becomes `75`).
Free text	`string`	
Regex	`string`	

Compatible response types for parameters

Arrays

In addition to the supported customer response types, a variable can also contain an array value. If you need to pass an array parameter to an operation, you must create an array session variable:

1. Create a new session variable, either using **Set variable values** in the step editor or from the **Variables > Created by you** page. (For more information about how to create a session variable, see Creating a session variable .)
2. In the **Type** field, select **Any**.
3. In the **Initial value** field, click the **Use expression** toggle to enable it. Enter an expression that defines an array value (such as `["New York", "London", "Tokyo"]`, `[123, 456, 789]`, or `[]`).

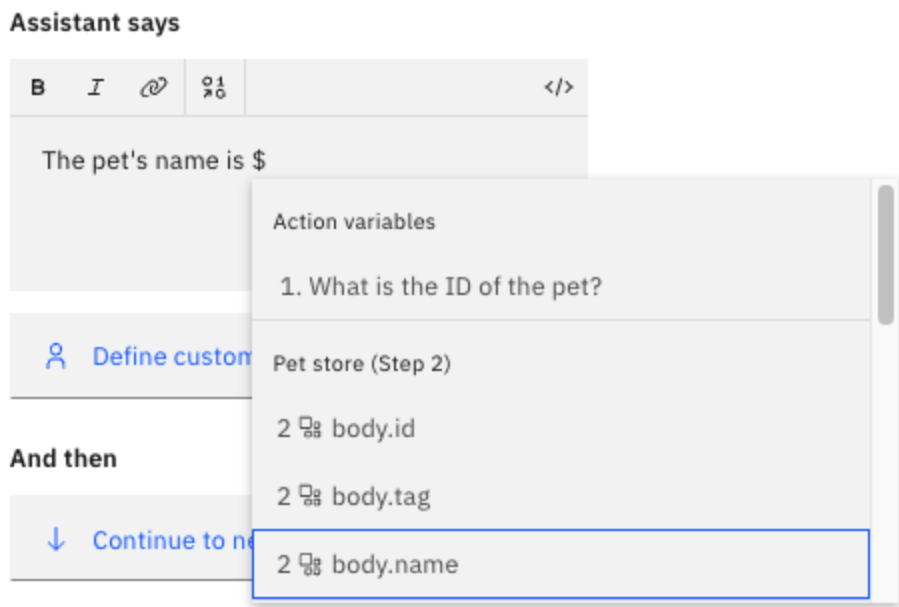
> **Note:** Because this variable contains an array value, your actions can use expressions with array methods to access or modify the array values. For example, you might want to create a variable that initially contains an empty array (`[]`) and then use the `add()` method to build a list one element at a time. For more information about the array methods you can use in expressions, see Array methods.

You can now select this variable as the value for a parameter that requires an array.

Accessing extension response data

After you call an extension, values from the response data are stored in special action variables that you can access in subsequent steps.

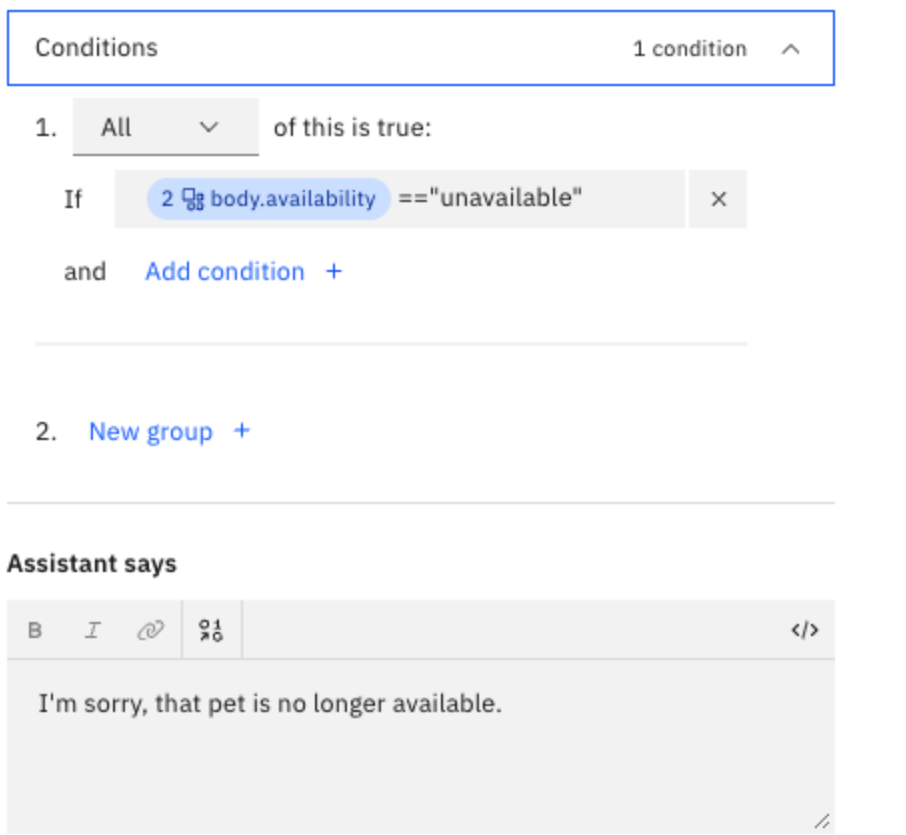
You can access these variables in the same way you access other action variables. You can reference it in the **Assistant says** text, evaluate it as part of a step condition, or assign it to a session variable so other actions can access it. The response variables are shown in the list of available variables, categorized under the name of the extension and the step from which it was called:



Each call to an extension creates a separate set of response variables. If your action calls the same extension multiple times from different steps, make sure you select the variables from the correct step.

Each variable represents a value from the response body. To make it easy to access these values, data is extracted from complex, nested objects and mapped to individual response variables. The name of each variable reflects its location within the response body (for example, `body.name` or `body.customer.address.zipcode`).

For example, this action step uses an expression to check the `availability` property in an extension response:



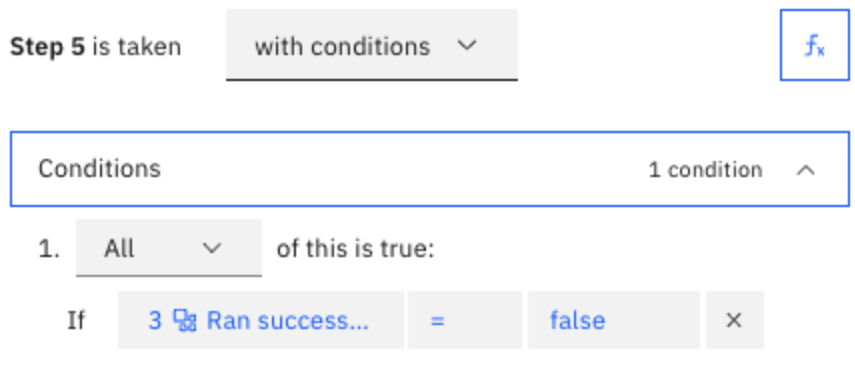
If a response variable contains an array, you can write an expression that uses array methods to access the elements of the array. For example, you might use the `contains()` method in a step condition to test whether the array contains a particular value, or the `join()` method to format data from the array as a string you can include in an assistant response. For more information about array methods, see Array methods.

Checking success or failure

You might want your assistant to be able to handle errors that occur when calling a custom extension. You can do this by checking the `Ran successfully` response variable that is returned along with the response from the call to the extension. This variable is a boolean (`true` or `false`) value.

If you define step conditions that check the `Ran successfully` variable, you can create steps that enable your assistant to respond differently depending on whether the call to the extension succeeded. (For more information about step conditions, see Step conditions.)

The following example shows a step condition that checks for a failure from an extension in step 3. By using this condition, you can create a step that tells the customer there was an error, and perhaps offers to connect to an agent for more help.



Conditioning on HTTP status

In addition to the `Ran successfully` variable, you might also want to create a step condition based on the HTTP status of the response. By doing this, you can create steps that handle the situation differently depending on the cause of the failure. For example, if the call failed because of a timeout error (HTTP status 408), you might want to retry the call.

> ⚠ **Important:** There are many possible HTTP status codes, and different methods use different status codes to indicate various types of success or failure. To condition on the HTTP status, you need to know what HTTP status codes the external service returns, and under what circumstances. These status codes are typically specified in the OpenAPI document that describes the external API.

To create an step condition based on the HTTP status code, follow these steps:

1. For the value you want to test, click **Expression**.
2. In the expression field, type a dollar sign (`$`) to show the list of available variables.
3. Select any variable that is a response value from the extension. (It doesn't matter which variable you select, as long as it is an extension response variable).

   The expression is automatically updated to show a reference to the variable you selected, in the format `${step_xxx_result_y.body.variablename}`. For example, if you selected a response variable called `body.id`, the reference might be `${step_596_result_1.body.id}`.
4. Inside the curly braces, (`{}`), edit this reference to remove `.body.variablename`. You should be left with something like `${step_596_result_1}`.
5. After the closing curly brace (`}`), add `.status`. The resulting reference identifies the status code returned from the call to the extension (for example, `${step_596_result_1}.status`).

   For more information about writing expressions, see Writing expressions.
6. Complete the expression by adding an operator and a comparison value, so the expression evaluates to a Boolean (true/false) value. For example, the following expression tests for HTTP status 408, which indicates a timeout error:

   ```
   $ ${step_549_result_1}.status==408
   ```

Debugging failures for custom extension

If your calls to an extension are failing, you might want to debug the problem by seeing detailed information about what is being sent to and returned from the system API. To do this, you can use the **Inspector** in the Preview pane:

1. Go to the Actions page, or the action editor, and click **Preview** to open the Preview pane.

   > **Note:** You cannot access the **Inspector** from the Assistant preview on the **Preview** page, which shows only what a customer would see. Instead, use the preview feature that is part of the Actions page, which gives you access to additional information.
2. Interact with your Assistant as a customer would.
3. Each time an extension is called, the preview pane shows a message giving you access to detailed information:

   Click **Inspect** to see details about the call to the extension.

   > **Tip:** You can also click the icon to show or hide the **Inspector**. However, you must click **Inspect** in the preview pane to show information about a particular call to an extension.

   The **Overview** tab of the **Inspector** shows the following information about a call to an extension:

Debugging option	Description
Extension	The name of the extension, as specified in the extension settings.
Operation	The operation that was called.
Status	The HTTP status code from the response. This code can help you determine if an error is being returned from the external service.

Request parameters	The input parameters that were sent to the system API as part of the request.
Response properties	The values of all properties included in the response from the system API. These are the values that are mapped to action variables after the call to the extension completes.

Tip: In the **Request parameters** and **Response properties** tables, long property names might be truncated to show only the last part of the JSON path. To see the complete path and property name, hover the mouse pointer over the property name in the table.

4. Click the **Advanced** tab in the extension inspector if you want to see the raw request and response data:
   - The request is shown as a cURL command, which you can run at a command prompt or import into a tool such as Postman. (For security reasons, the content of any `Authorization` header is not included.)
   - The response is shown as the complete JSON data returned from the system API.

Debugging failures for Conversational search or skill based actions

If your calls to the Conversational search or skill-based actions fail, you might want to debug the problem by seeing the detailed information about what is being sent to and returned from the system API.

Note: The conversational search inspector shows up only when conversational search is enabled in your search integration. If you are using Custom service search integration, you must use only server-side search when you configure the search integration. The client-side search is not supported in conversational search inspector.

To see the detailed information for analyzing the problem, use the **Inspector** in the Preview pane:

1. Go to the Actions page, or in the action editor, click **Preview** to open the Preview pane.

   Note: You cannot access the **Inspector** from the Assistant preview on the Preview page. Instead, use the preview feature that is part of the Actions page, which gives you access to additional information.

2. Interact with your Assistant as a customer would.
3. Each time an extension is called, the preview pane shows a message to access the detailed information: You can also click the icon to show or hide the extension inspector. Click **Inspect** in the preview pane to show information about a particular call to the search integration.
4. Use the **Overview** tab to find the reasons for the failure of your calls to Conversational search.

   Understand the two phases of the search extension before knowing about the fields that are displayed in the **Overview** tab.

   - Retrieval phase

     Represents the initial search phase where an external document search engine is called to retrieve the initial set of results.

   - Answer generation phase

     Represents the phase where the data is retrieved during the retrieval phase and are sent to an LLM to generate a human readable answer for the user.

The Overview tab of the Inspector shows the following information about the call to the search integration.

Debugging option	Description
Extension	The name of the extension, as specified in the extension settings.

Index	The name of the Elasticsearch index used by the search, visible only when search extension is configured to use Elasticsearch.
Project Id	The ID of the project used by IBM Watson® Discovery during the retrieval phase of search. This field is visible only when you configure search extension to use IBM Watson® Discovery.
Query	The query used by the system to initiate search on the document engine (Elasticsearch, IBM Watson® Discovery, or Custom service server-side). The value of this field reflects the system's rewritten query.
Original Query	The query through which the user initiated the search. This field is visible only when the system rewrites the query when multi-turn conversational search is enabled.
Custom results filter	Shows the custom results filter, if provided on the search trigger, that triggered the conversational search. This field might not appear in responses always.
LLM type	The LLM that was called during the answer generation phase. This value is watsonx.ai. To know more about configuring LLM in your Assistant, see Base LLM.
Model	The model used by the base LLM during the answer generation phase of the search.
Stream close reason	Gives the reason why the stream ended or was closed with a corresponding value in the UI. This field is visible only when you enable streaming in the Web Chat.
LLM input token count	Gives the number of tokens in the request that was sent to the LLM in the answer generation phase of the search.
LLM generated token count	Gives the number of tokens in the answer that the LLM responds with, in the answer generation phase of the search.
IDK response	This field is visible only when the search answer resolves to an IDK (I don't know) response.
IDK reason	A reason for why a search answer was resolved to an IDK (I don't know) response by the system is shown with the corresponding value.
IDK trigger opening phrase	The field shows the detected standard opening phrase that triggered an IDK (I don't know) response, visible only when the IDK reason is due to this phrase.
IDK trigger phrase	The field displays the detected trigger phrase that caused an IDK (I don't know) response, visible only when the IDK reason is due to this phrase.
Total time to return	The total time the system took to complete the execution of Conversational search.
Search time	The time taken for the system to call the search engine and retrieve results in the retrieval phase of the search.
Answer generation time	The time taken for the system to complete the answer generation phase of the search.
Confidence threshold	A numerical value that represents the search confidence threshold, also known as Tendency to say "I don't know" in the Assistant configuration.
Extractiveness	The score reflects how much of the response is directly quoted from sources. A higher score indicates more direct quoting and a lower score indicates rephrasing of the source or lack of source support.
Retrieval confidence	The confidence value indicates how sure the system is about the search results. If it's below the threshold, the system responds with IDK and the reason Retrieval confidence too low.

Response confidence	The confidence value indicates the system's certainty in view of the generated answer. If it's below the threshold, the system responds with IDK and the reason `Response confidence too low`.

Reconfiguring a missing extension

An extension might become unavailable if someone removes it from the assistant on the **Integrations** page, or if the action is exported and then imported to a different assistant where the required extension is not configured. If this happens, any action step that calls the extension becomes invalid.

To correct the problem, follow these steps:

1. If necessary, recreate the extension using the same OpenAPI specification that was used before. (For more information, see Building a custom extension.)
2. Make sure the extension has been added to the assistant. (For more information, see Adding an extension to your assistant.)
3. In the action editor, edit the action step that calls the extension and check whether the call to the extension is correctly configured. If watsonx Assistant recognizes the required extension, the extension configuration is automatically restored.

   If you see the message `Extension not fully configured`, this means that watsonx Assistant did not find the required extension. Click **Edit extension**.
4. In the **Extension setup** window, select the extension you want to call.

   > **Note:** If watsonx Assistant recognizes an available extension that was built using the same OpenAPI document, a message appears suggesting that you select this extension. However, you can select any available extension.

5. Verify that the correct values are specified in the **Operation** and **Parameters** fields.
6. Click **Apply**.
7. If you selected an extension that is not identical to the one that was used to build the action, you might need to modify subsequent steps that access the extension response properties. Check any later steps that refer to the response properties and make sure the references are still valid and correct.

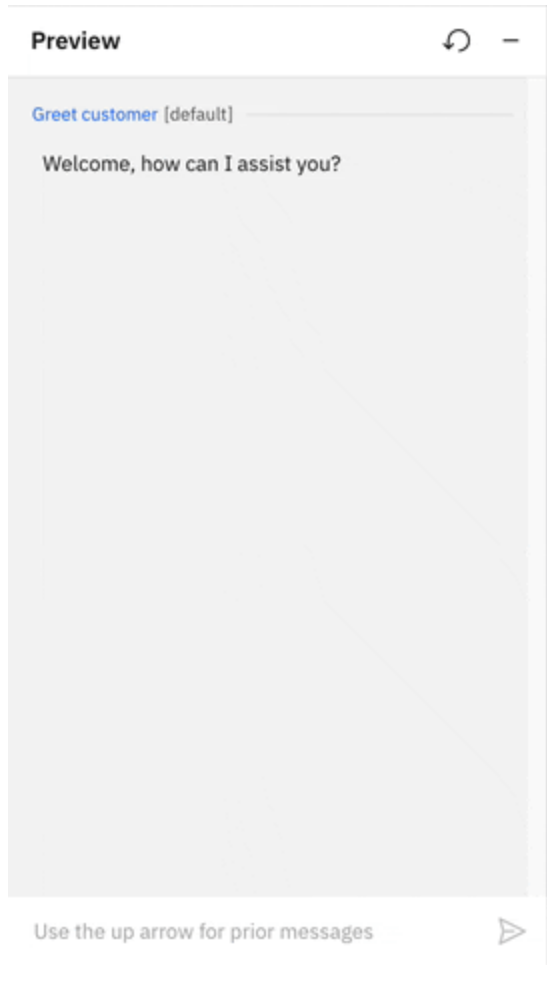
Allowing your customers to change the topic of the conversation

In general, an action is designed to lead a customer through a particular process without any interruptions. However, real conversations almost never follow such a simple flow. In the middle of a conversation, customers might get distracted, ask questions about related issues, misunderstand something, or change their minds about what they want to do.

The **Change conversation topic** feature enables your assistant to handle these digressions, dynamically responding to the user by changing the conversation topic as needed.

How changing the topic works

This example shows a customer who changes the conversation topic while they are entering a credit card. When the assistant asks `What is your CVV number?`, the customer (who is new to credit cards) asks what a CVV is. The assistant is designed to handle this possibility, so it switches to a different action that answers the customer's question. It then continues where it left off.



The assistant determines when to change the conversation topic as follows:

1. When the assistant asks a question and receives a response, it first validates the response to see whether it answers the question. In the CVV example, the assistant expects a number as a response.
2. If the input is not recognized as an answer to the question, the assistant then evaluates the input to see whether it matches another action. (Changing the topic cannot switch the conversation to a different assistant.)

   Note: If the input does not answer the question, and it does not match any existing action of the assistant, a step validation error results. For more information about validation errors and how they are handled, see Handling errors in the conversation.
3. If the input matches a different action, the assistant switches to the matching action.

   You can control how often your assistant switches topics. For more information, see Setting threshold.

   In the example, the customer's response (`What's a CVV number?`) is not a valid response, but it does match another action that is designed to answer this question. The matching action is triggered, answering the customer's question.
4. After the second action completes, the assistant asks the customer if they want to return to the original action. If they say `Yes`, the assistant continues with the step where the customer changed the topic. In the example, the assistant returns to the original action and repeats the question `What is your CVV number?` .

Note: If an action starts and doesn't finish its step set to *End the action*, then a customer can't digress into that action while it is still in progress. For example, if a customer starts an action, changes the topic to start another action, and then changes the topic again, any in-progress action isn't available for a topic switch.

Enabling and disabling changing the topic

By default, the **Change conversation topic** feature is enabled for all assistants and actions. (You need to enable the feature for steps that use a free text or regex response. For more information, see Enabling changing the topic for free text and regex customer responses .)

However, some processes are best completed without interruption, so you might want to disable this feature. You can disable for all actions, or just for an individual action.

To disable changing the topic for all actions:

1. From the **Actions** page of the assistant, click **Global settings** .
2. On the **Change conversation topic** tab, set the switch to **Off**.
3. Click **Save**, and then click **Close**.

To disable changing the topic for an individual action:

1. Edit an action, then click **Action settings** .

2. In the Action Settings window, toggle the **Change conversation topic** switch to **Off**.

Disabling returning to the original topic

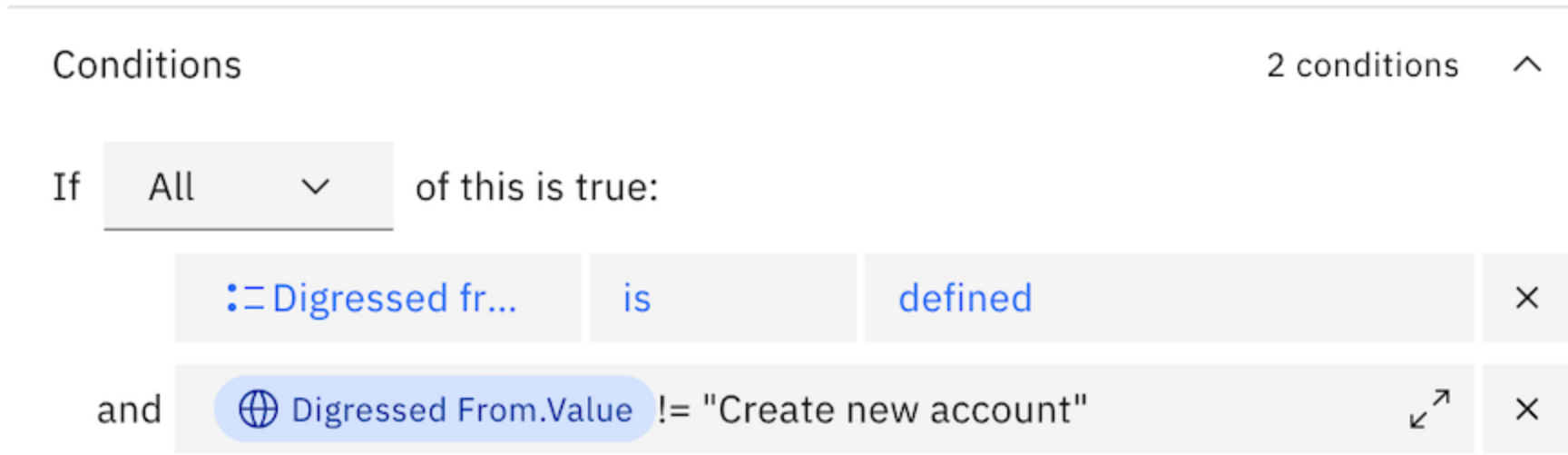
Even when you allow changing the topic, you might not want a customer to return to the previous topic.

To disable returning to the original topic:

1. Edit an action, then click **Action settings** ⚙.
2. Keep the **Change conversation topic** toggle switched to **On**.
3. Select the checkbox **Never return to original action after completing this action** to prevent the customer from returning to the previous action.

Conditioning on previous topic

You can condition your new topic action, or steps within your new topic action, on the previous topic by using the built-in variable `Digressed from`. For example, if your previous topic action was `Create new account` and you digressed to an action with steps that only apply to existing accounts, use the expression in the following image in your step conditions to decide to run those steps.



For more information on creating conditions, see Adding conditions to an action and Adding conditions to a step.

Allow change of topic between actions and dialog

If you are using actions and dialog, you can ensure that customers can change topics between an action and a dialog node.

Note: This setting is available if you activate dialog in Assistant settings. For more information, see Activating dialog and migrating skills.

1. Set the toggle **Change topics from actions to dialog** to **On**.
2. Click **Save**, and then click **Close**.

Enabling changing the topic for free text and regex customer responses

By default, changing the conversation topic works differently for free text and regex responses. Customers can't change topics when:

- The assistant is asking for a free text response
- An utterance matches the pattern in a regex response

If you want a customer to be able to digress and change topics when they enter a free text or regex answer:

1. Within the step, click the **Settings** icon for the customer response.
2. Enable the toggle **Allow customer to change topics during a free text response** or **Allow customer to change topics before evaluating a regex response**. For more information, see Free text or Regex.

Confirmation to return to previous topic

By default, new assistants are set to ask a `yes or no` question to confirm that the customers want to return to the previous action. You can modify these settings.

Note: These settings aren't available when the assistant language is set to `Another language`, which uses the universal language model. For more information, see Adding support for global audiences.

To change the confirmation settings:

1. From the **Actions** page of the assistant, click **Global settings** ⚙.
2. On the **Change conversation topic** tab, click the **Confirmation** section.

Settings that you can change are:

Setting	Description
Ask a confirmation question	Change the toggle to **Off** to disable the confirmation question for all actions.
Assistant says	By default, the assistant asks `Do you want to return to your previous topic?` and uses the system variable `Digressed from` to include the name of the previous action. Use the editor to modify the confirmation question.
Validation message	By default, the assistant says `Please reply back with yes or no` if the customer doesn't answer `Yes` or `No`. Use the editor to modify the validation message.
Show confirmation buttons	The assistant includes clickable `Yes` and `No` buttons with the confirmation question. Click the toggle to **Off** to disable the `Yes` and `No` buttons.

Confirmation settings

Providing options when a question or request cannot be answered

No matter how well your assistant is designed, customers can sometimes encounter issues with the chatbot understanding them or doing what they want. Your assistant can automatically help customers recover from many situations, and you can configure how it responds.

How error conditions are handled

Error situations that your assistant can recover from:

- Your assistant cannot understand your customer's request.
- Your customer does not provide a valid response to a question.
- Your customer requests to speak to a live agent.

Your assistant can detect error conditions and give customers the chance to correct them. In addition, the built-in *Fallback* action provides a way to automatically connect customers to a live agent if they need more help.

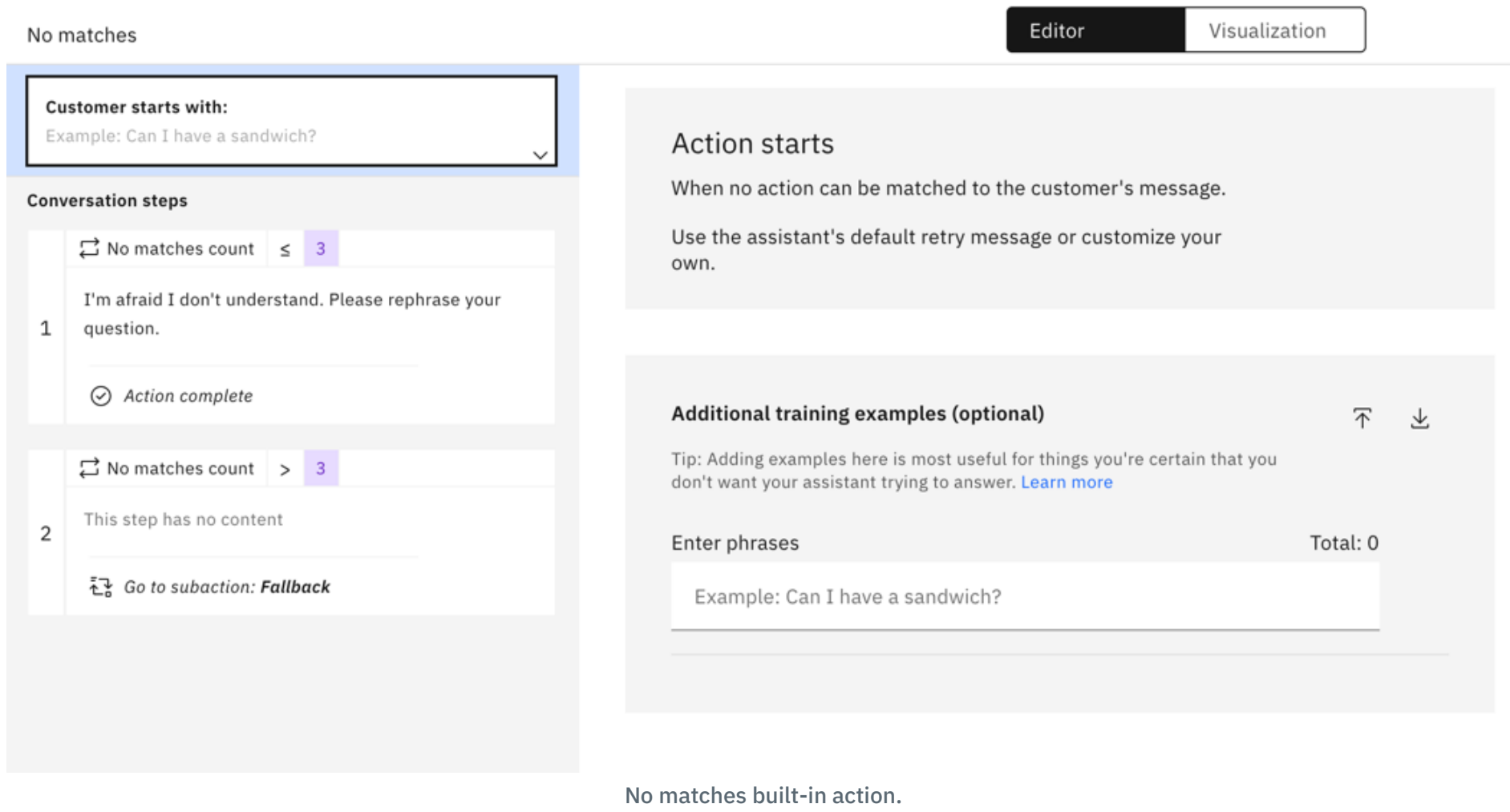
When the assistant can't understand your customer's request

When you build actions, you train your assistant on what your customers might ask for. The **Customer starts with** section of each action provides examples of customer input that trigger the action. The assistant uses natural language processing to recognize customer input that is similar to these examples. This happens at the beginning of the conversation, or after an action completes and the assistant is ready for another action.

You cannot anticipate every possible request, so sometimes customers send input that your assistant fails to match to any action. The input may be phrased in a way that the assistant cannot understand, or customers ask for things that your assistant is not designed to handle.

You can configure your action steps to route to search or **No matches** if the user input do not match any action. For more information about configuring search routing, see Configuring the search routing when no action matches.

Unrecognized input of this sort triggers the built-in **No matches** action. To see how this action works, click **Set by assistant** in the list of actions, and then click **No matches**.



No matches built-in action.

By default, this action has two steps, and each step is conditioned on the *No matches count* session variable. This built-in variable is automatically incremented with each consecutive unrecognized input. Therefore, the behavior of the **No matches** action differs depending on how many times in a row the user says something that the assistant fails to understand.

- For the first three unrecognized messages, step 1 executes. This step outputs a message that the assistant did not recognize the user's input, and asks the user to try again. You can edit the message, or modify the number of times that the assistant responds with this message.

  When the user sends input that successfully triggers an action, the *No matches count* session variable is reset to 0.

- If the user tries more than three times and the assistant still does not understand, step 2 executes. Step 2 calls the *Fallback* action, which offers other options such as connecting to a live agent. For more information, see Editing the fallback action.

Tip: You can edit the **No matches** action just as you can any other action by changing the existing steps and adding or deleting steps. If you change the **No matches** action, you might accidentally break your assistant's ability to recover from errors in the conversation. If this happens, you can re-create the default steps.

Setting threshold

You can set how often customers are routed to **No matches** by changing a global setting for actions.

1. From the **Actions** page of the assistant, click **Global settings**.
2. Click **Conversation routing** tab.
3. Click **No matches** tab.
4. Drag the slider to set the threshold, or choose an option from the drop-down box.

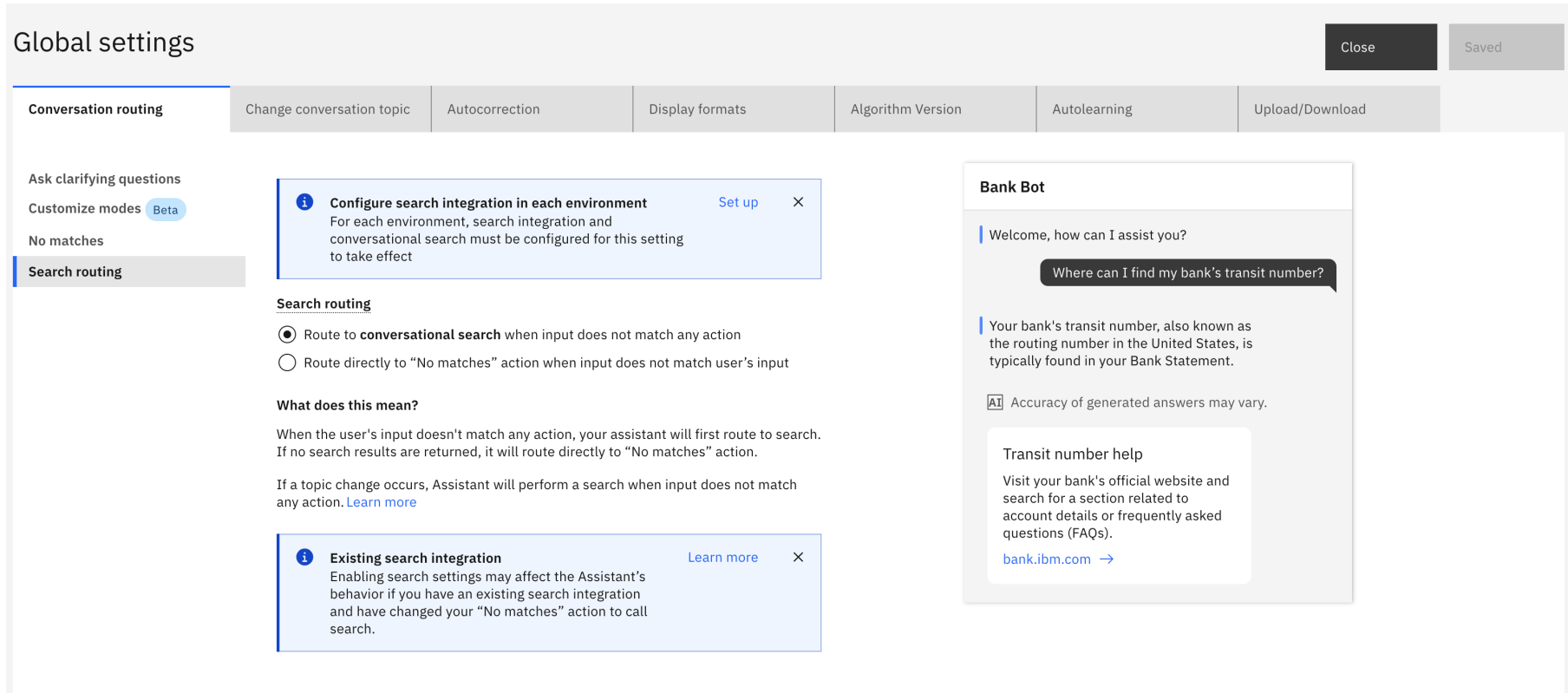
Configuring search routing when no action matches

You can configure your assistant's behavior for the case when a user's input does not match any actions by selecting one of the following options:

- **Route to conversational search**

  Select this option for your assistant to route to conversational search when the user's input does not match any actions. If conversational search generates a response that falls below its response confidence score threshold, the assistant falls back to your **No matches** action. You can configure the response confidence score threshold or **Tendency to say "I don't know"**, in the conversational search settings.

  Note: Before you select **Route to conversational search**, you must configure the search integration and enable the conversational search.



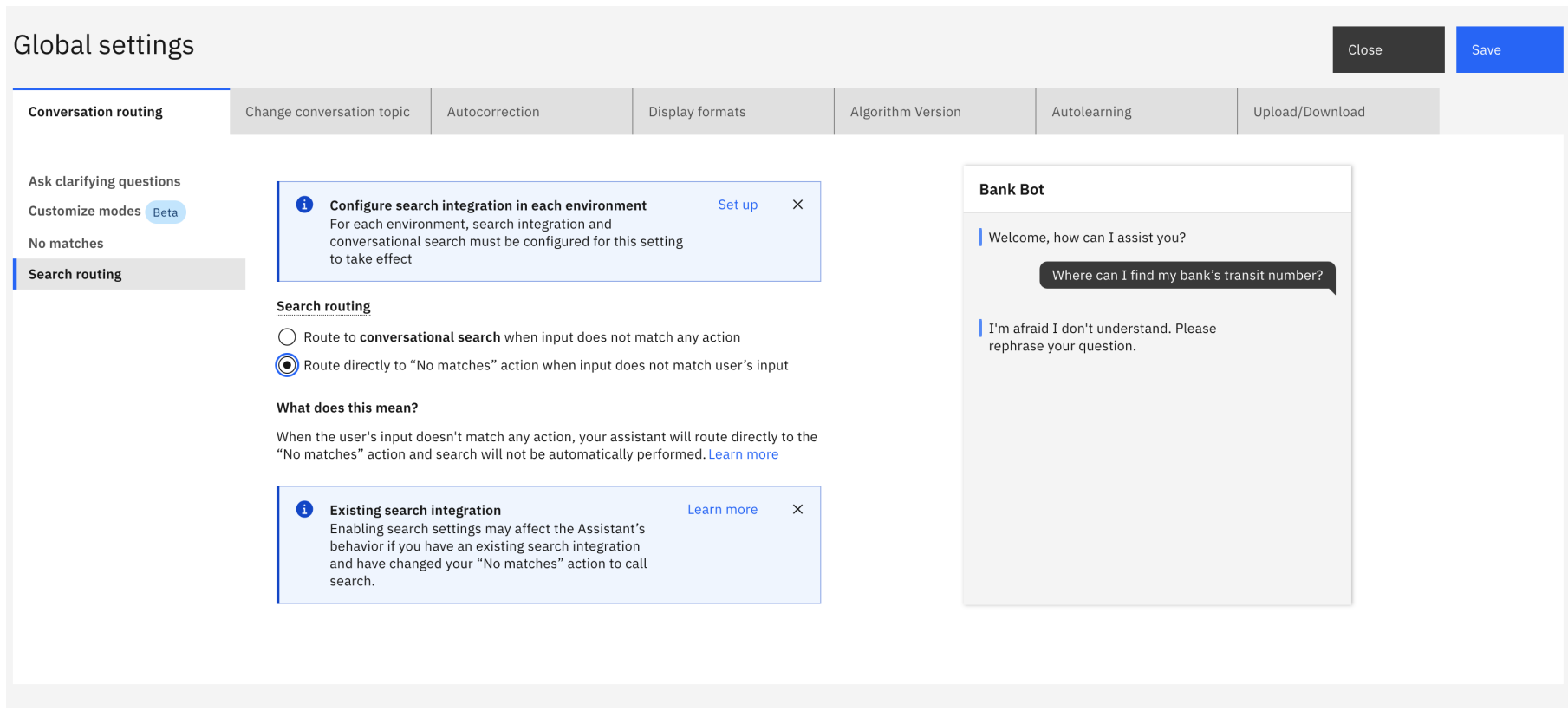
Route to conversational search

- **Route directly to the No matches action**

Select this option for your assistant to route directly to the **No matches** action when the user's input does not match any actions. You can customize the behavior of the **No matches** action per your use case. You can configure the **No matches** action to route to:

- Conversational search
- Transfer to a live agent
- Open a support ticket
- Understand your users' questions and requests

You can use this option to route the assistant users to the **No matches** action when there are no matches for the user responses.



Route directly to No matches action

To configure **Search routing**, go to **Home** > **Actions** > **Global settings** > **Conversation routing** > **Search routing**. After you select an option **Search routing**, click the **Save** button.

Tip: For any new assistant, the **Route to conversational search** option is the default selection in the **Search routing** section.

Tip: The **Search routing** feature is enabled for any assistant in draft environment, but not for assistants in live environment. However, the **Search routing** feature is enabled for the assistant in the live environment, if you created the assistant in live environment from a draft environment for which **Search routing** is configured earlier.

Adding examples of unsupported input

By default, the **No matches** action is triggered only when the user's input does not match any defined action.

If you can anticipate certain user requests that your assistant does not support, you can add these requests as example phrases in the **Customer starts with** section of the **No matches** action. Adding example phrases helps to ensure that these requests are sent directly to the **No matches** action rather than triggering a different action by mistake. You can also upload or download example phrases in a comma-separated value (CSV) file. For more information, see Adding more examples.

When your customer gives invalid answers

When a step in an action asks your customer to answer questions or provide additional information, the assistant expects a particular response type, such as a number, date, or text string. (For more information, see Collecting information from your customer.) The assistant checks the customer's response to ensure it fits the expected response type; this process is called *validation*.

For most customer response types, the assistant is able to understand valid responses in various formats. For example, for a time value, `2:15 PM` and `a quarter past two in the afternoon` are both acceptable. But if the user provides a value that the assistant cannot interpret as matching the expected response type (for example, a response of `purple` when asked for a number), a validation error results.

When a validation error occurs, the assistant asks the customer to try again. By default, the assistant allows three attempts for a valid response. After the third attempt, another invalid attempt triggers the *Fallback* action, which offers other options such as connecting to a live agent. For more information, see Editing the fallback action.

Customizing validation for a response

When you edit a step that expects a customer response, you can customize how validation errors are handled.

1. Click **Edit validation** to see the validation options.
2. For all customer responses except *free text*, you can customize the following options:
   - In the **Response 1** field, specify the text of the message the assistant sends when the customer's response does not match the expected response type. For example, the default validation error message for a numeric value is `I didn't catch that. Enter a number.` You might want to customize this message to be more specific, such as `Enter the number of people in your group.`
   - If you want to use several validation responses, click **Add Response** to add more validation messages. The number of validation responses you can enter depends upon the value in the **If attempts exceed** field.
   - In the **If attempts exceed** field, click `+` or `-`, or directly edit the number, to change how many consecutive attempts the customer can make before the *Fallback* action is triggered. Or, if you enabled response modes, you can use the number of step validation attempts from the response mode that you are using.
   - Select **Repeat Assistant Says response** if you want the customer to see or hear the original question. For example, following a validation message of `I'm sorry, please choose a day when we are open`, the assistant repeats `When do you want to visit? We are open Monday through Friday`.

For date, time, and numeric customer responses, you can customize the validation to check for a specific answer, such as a range of dates or a limited currency amount. Each choice is optional so you can build a validation specific to the response.

Response type	Validation choices
Number	Minimum, maximum
Date	After date, before date, specific days of the week
Time	Start time, end time
Currency	Minimum, maximum
Percentage	Minimum, maximum

Validation choices

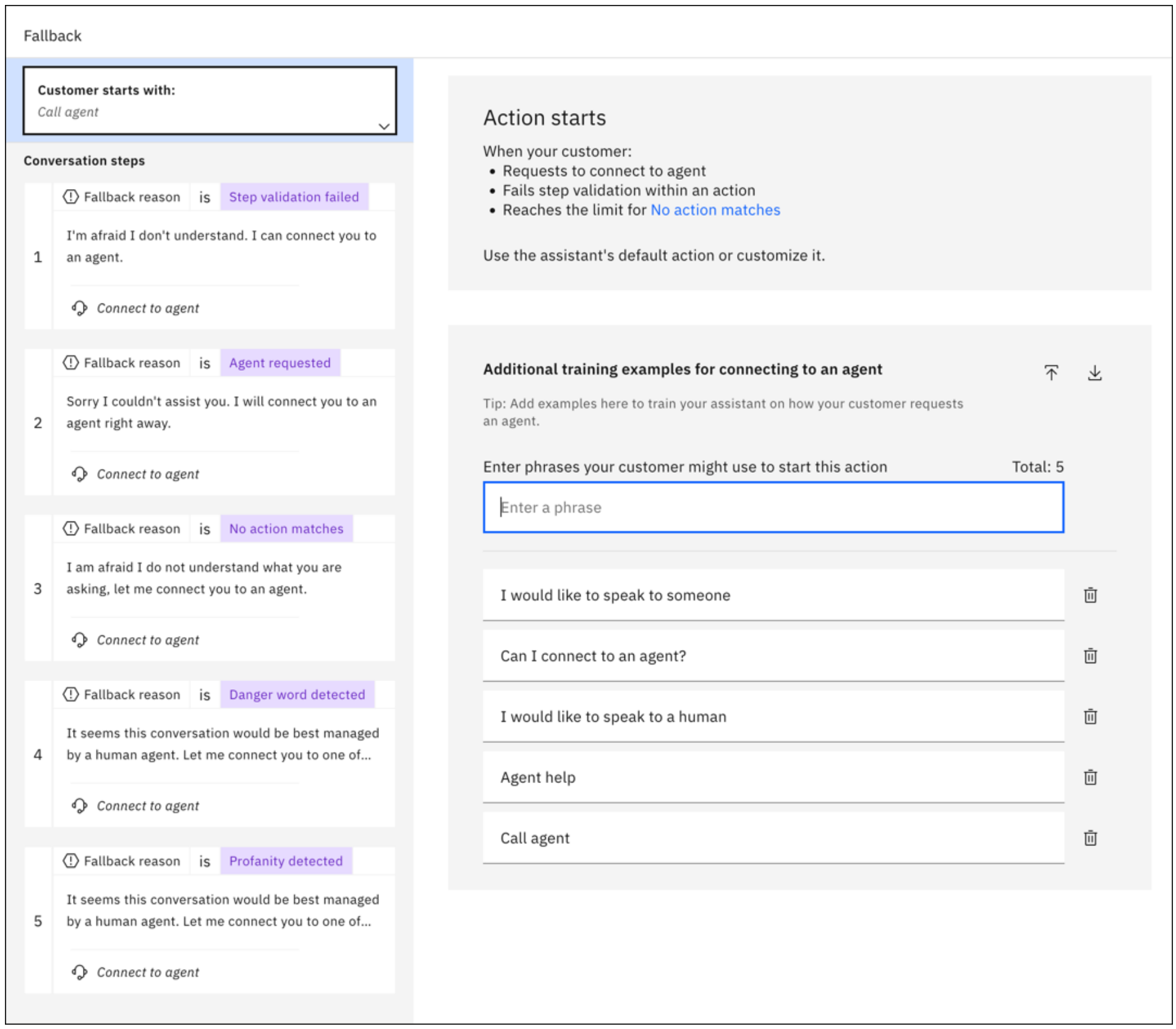
When your customer asks to speak to a live agent

At any point in the conversation, your customer might ask to speak to a live agent. The built-in *Fallback* action is preconfigured with example input that detects such requests; you can edit the **Customer starts with** section of the *Fallback* action to add more examples. You can also upload or download example phrases in a comma-separated value (CSV) file. For more information, see Adding more examples.

Editing the fallback action

The built-in *Fallback* action is automatically provided with each assistant and cannot be deleted. However, you can edit the *Fallback* action to modify the conversation that your users have with the assistant when errors occur. For example, you might want to add steps or modify step conditions to provide more control of how specific error conditions are handled.

To edit the *Fallback* action, click **Set by assistant** in the list of actions, and then click *Fallback*.



Fallback built-in action

Whenever the *Fallback* action is triggered, the assistant also sets a value for the *Fallback reason* session variable. This value indicates what caused the *Fallback* action to be triggered. By default, this variable can have one of five values:

- *Step validation failed*: The customer repeatedly replied with invalid answers for the expected customer response type.
- *Agent requested*: The customer directly asked to be connected to a live agent.
- **No matches**: The customer repeatedly made requests or asked questions that the assistant did not understand.
- *Danger word detected*: The customer uses words or phrases that match the *Connect to agent* step in the *Trigger word detected* action.
- *Profanity detected*: The customer repeatedly used words or phrases that match the *Show warning* step in the *Trigger word detected* action.

For more information about the *Trigger word detected* action, see Detecting trigger words.

The *Fallback* action defines five conditional steps, one for each possible value of the *Fallback reason* variable. Each step sends a message to the customer, based on the error condition, and then uses the *Connect to agent* feature to transfer the conversation to a live agent. (For more information about this feature, see Connecting to a live agent.) You can modify these steps if you want to handle an error condition in a different way.

Connecting to a live agent

Your assistant can do a lot, but there might be situations when customers need help from a human. If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers the conversation to a live agent when necessary. This is referred to as an *escalation*.

To use this feature, your assistant must interact with customers using a **web chat** or **phone** integration. For more information, see Deploying your

assistant.

You must add an integration to a service desk system before setting up live agent transfers to your assistant. The supported service desks depend upon how customers connect to your assistant (web chat or phone). For more information about integrating with a service desk, see Basic web chat configuration and Phone integration configuration.

Your agents can work with one of these supported service desk tools:

- Genesys
- NICE CXone
- Salesforce
- Twilio Flex
- Zendesk
- Bring your own - starter kit and Bring your own service desk

There are two basic scenarios when your assistant might need to transfer a conversation to a live agent:

- *Planned escalation* refers to any anticipated situations in which you know you always want to hand off the conversation to a live agent.
- *Fallback escalation* is an unexpected situation in which the customer is unable to get help from the assistant.

When the assistant initiates a transfer, the agent receives a notification within the agent dashboard, and has access to the history of the customer's chat with the assistant.

Planned escalations

Examples of planned escalations might include the following:

- The customer asks for a service that cannot be completed without the assistance of a live agent.
- The customer needs help with a sensitive subject that requires a human touch, such as asking about bereavement benefits or resolving a complaint.

To set up a planned escalation, you build an action that can recognize a specific situation that requires a live agent. An example would be an action that is triggered by customer input `I want to pay my bill` (you might want to let live agents handle payments for security reasons).

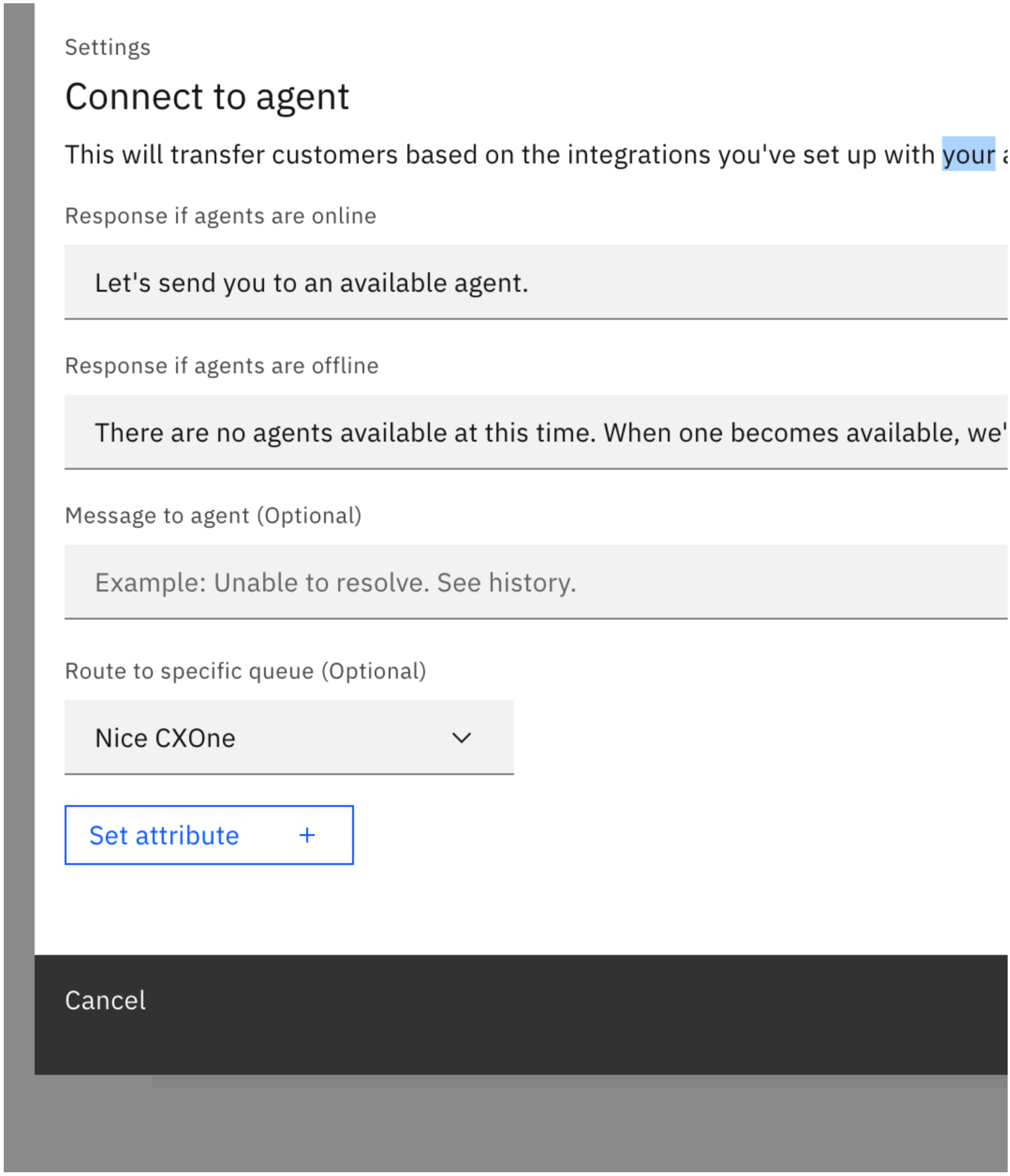
Within any action, you can create a step that initiates a transfer to a live agent:

1. Add a step or edit an existing step to transfer the conversation to a live agent.

   Transferring the conversation to a live agent ends the action. To continue the conversation within the assistant instead of transferring, use step conditions as needed.

> **Note:** The action completes once the transfer occurs, regardless of whether the customer is successfully engaged with a live agent after the transfer.

1. In the **And then** field at the end of the step, select **Connect to agent**.
2. In the **Settings** window, you can customize messages the assistant displays as part of the transfer:



- **Response if agents are online**: The message the assistant sends to the customer when the conversation is being transferred to an agent. The default message is `Let's send you to an available agent`.
- **Response if agents are offline**: The message the assistant sends to the customer when no agents are currently available to take over the conversation. The default message is `There are no agents available at this time. When one becomes available, we'll connect you.`
- **Message to agent**: An optional message the assistant sends to the live agent when transferring the conversation.
- **Route to a specific queue**: An optional selection to route customers to a specific integration, which can be helpful if you have more than one set up.

3. Click **Apply**.

> **Note:** If you want to edit the transfer settings later, click **Edit settings** in the **And then** field.

Fallback escalations

Examples of fallback escalations include:

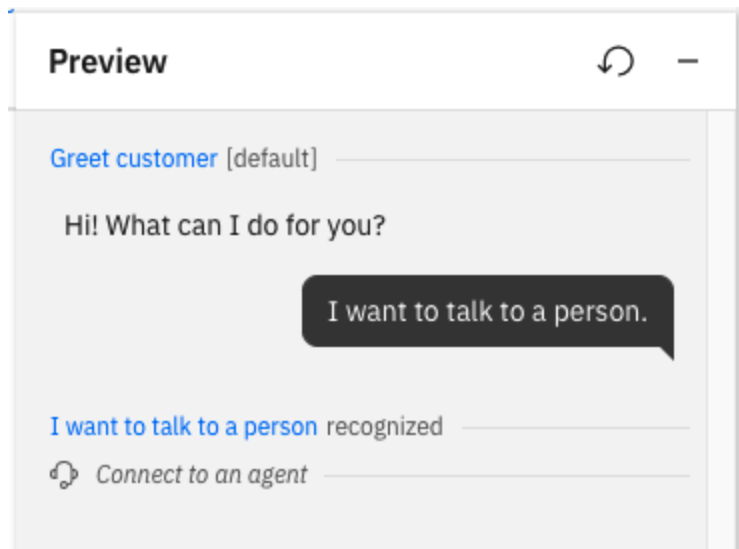
- The customer repeatedly asks a question or makes a request that the assistant cannot match to any defined action.
- The customer repeatedly gives an invalid answer to a question.
- The customer explicitly asks to speak to a human.

Fallback escalations use the *Fallback* action, which is a built-in system action that is automatically triggered in any of these fallback scenarios. By default, the *Fallback* action handles these error conditions by initiating a transfer to a live agent.

For more information about this automatic error handling and the *Fallback* action, see Handling errors in the conversation.

Testing the transfer in the Preview pane

After you have configured an action to connect to a live agent, you can preview it by clicking **Preview**. Note that in the action Preview pane, no actual transfer takes place, but the *Connect to an agent* message confirms that it was correctly triggered.



If you want to test the actual transfer using a working service desk integration, you can do so using the assistant Preview page. For more information, see Previewing and sharing your assistant.

Detecting trigger words

IBM Cloud

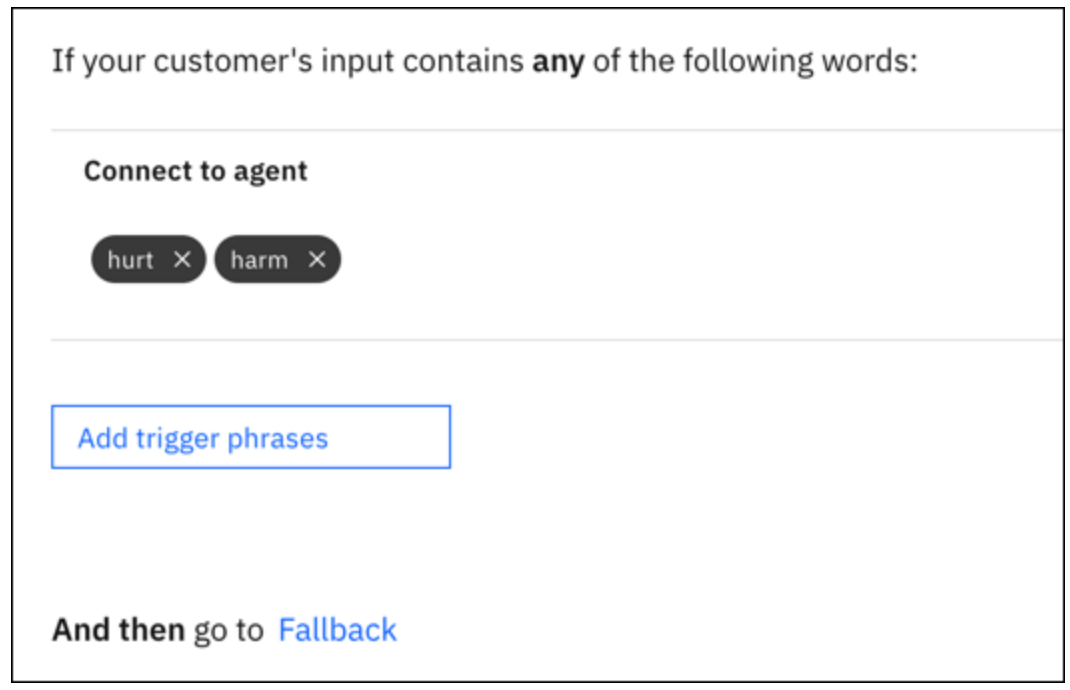
Use the *Trigger word detected* action to add words or phrases to two separate groups. The first group connects customers with an agent. The second group shows customers a customizable warning message.

By default, this action has two steps: the *Connect to agent* step and the *Show warning* step. To see how this action works, click **Set by assistant** in the list of actions, and then click **Trigger word detected**.

Connect to agent

The first step of the *Trigger word detected* action is the *Connect to agent* step. The *Connect to agent* step goes to the *Fallback* action if any trigger words are detected in the customer's input. Use this step to capture any key phrases where it's important to connect a customer with a live agent rather than activate any further actions.

For example, you might add `hurt` and `harm` as trigger words for the *Connect to agent* step:



Adding trigger words to the Connect to agent step

In this example, a customer enters a word or phrase that includes `hurt` or `harm`, which triggers the *Fallback* action. Step 4 has:

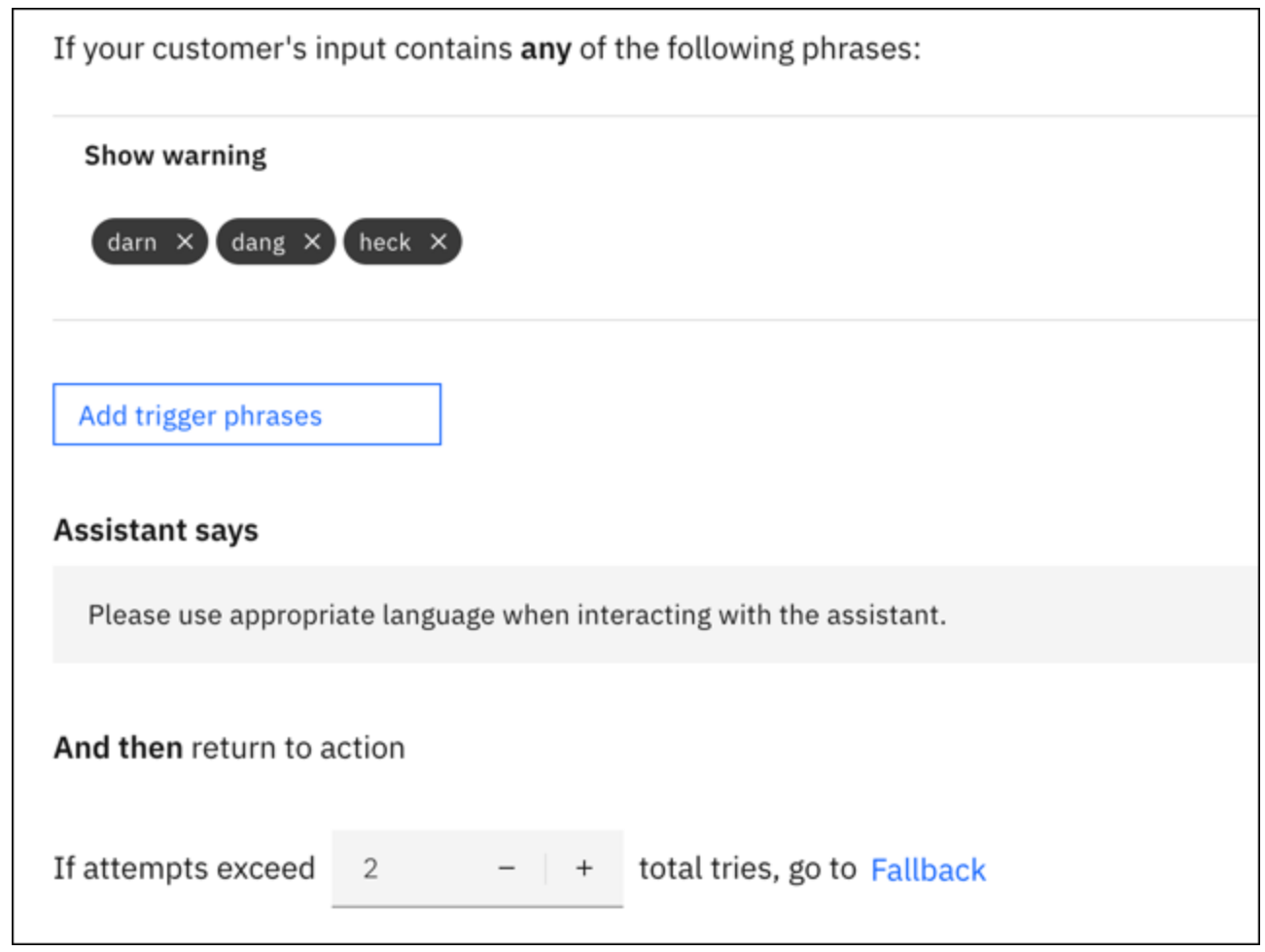
- `Danger word detected` as the fallback reason.
- The default message: `It seems this conversation would be best managed by a human agent. Let me connect you to one of our agents.` You can customize this message.
- **And then** set to **Connect to agent (action ends)**.

For more information about the *Fallback* action, see Editing the fallback action.

Show warning

The second and final step of the *Trigger word detected* action is the *Show warning* step. The *Show warning* step shows a customizable warning message to your customer if any trigger words are detected in the customer's input. Use this step to discourage customers from interacting with your assistant in unacceptable ways, such as using profanity.

For example, you might add `darn`, `dang`, and `heck` as trigger words for the *Show warning* step:



Adding trigger words to the Show warning step

In this example, a customer enters `darn`, `dang`, or `heck`, the assistant responds with `Please use appropriate language when interacting with the assistant`. You can customize this message.

If the customer triggers the *Show warning* step again, the *Fallback* action is triggered. The default setting is if attempts exceed 2 total tries. You can customize this setting.

In the *Fallback* action, step 5 has:

- `Profanity detected` as the fallback reason.

- The default message: `It seems this conversation would be best managed by a human agent. Let me connect you to one of our agents.` You can customize this message.
- **And then** set to **Connect to agent (action ends)**.

For more information about the *Fallback* action, see Editing the fallback action.

Saving your actions

When working on actions, your assistant automatically saves changes when you do one of the following:

- Click on a new step.
- Open Preview.
- Reset Preview.

You can also save changes by clicking the **Save** icon, for example, in these places when working on actions:

- Within each action.
- Actions settings.
- Renaming an action.
- Editing a variable.

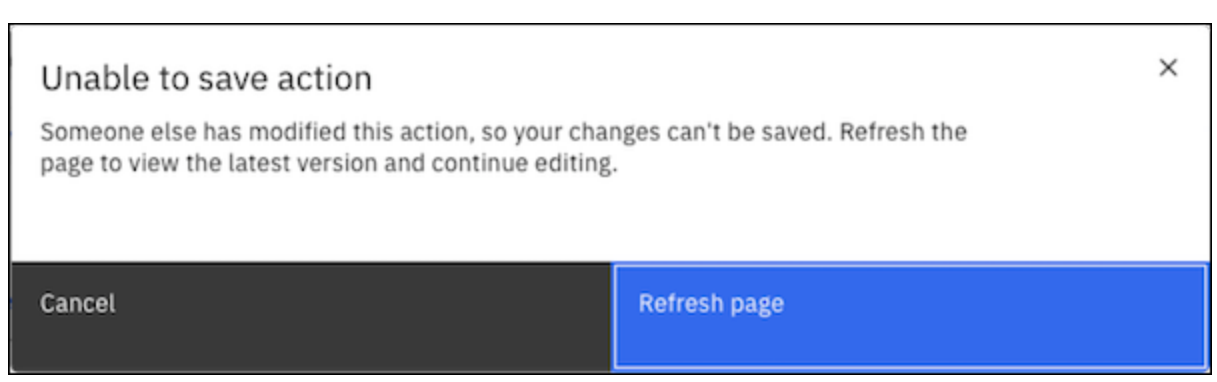
When the system automatically saves or you click the **Save** icon, the following items are saved to all:

- Actions that you created.
- Edits that you made to a system action.
- Variables that you created.
- Action settings.

Avoiding conflicts

To avoid conflicts when saving, we recommend dividing work so that one person works on one action at a time.

Also, watsonx Assistant prevents two users from working in an action with unsaved edits. If one user saves changes before a second user, this `Unable to save action` message might appear:



Reviewing and debugging your actions

Learn how to test the conversation you built into an action to experience what your users see with your assistant. If there are any issues, learn how to debug the user experience.

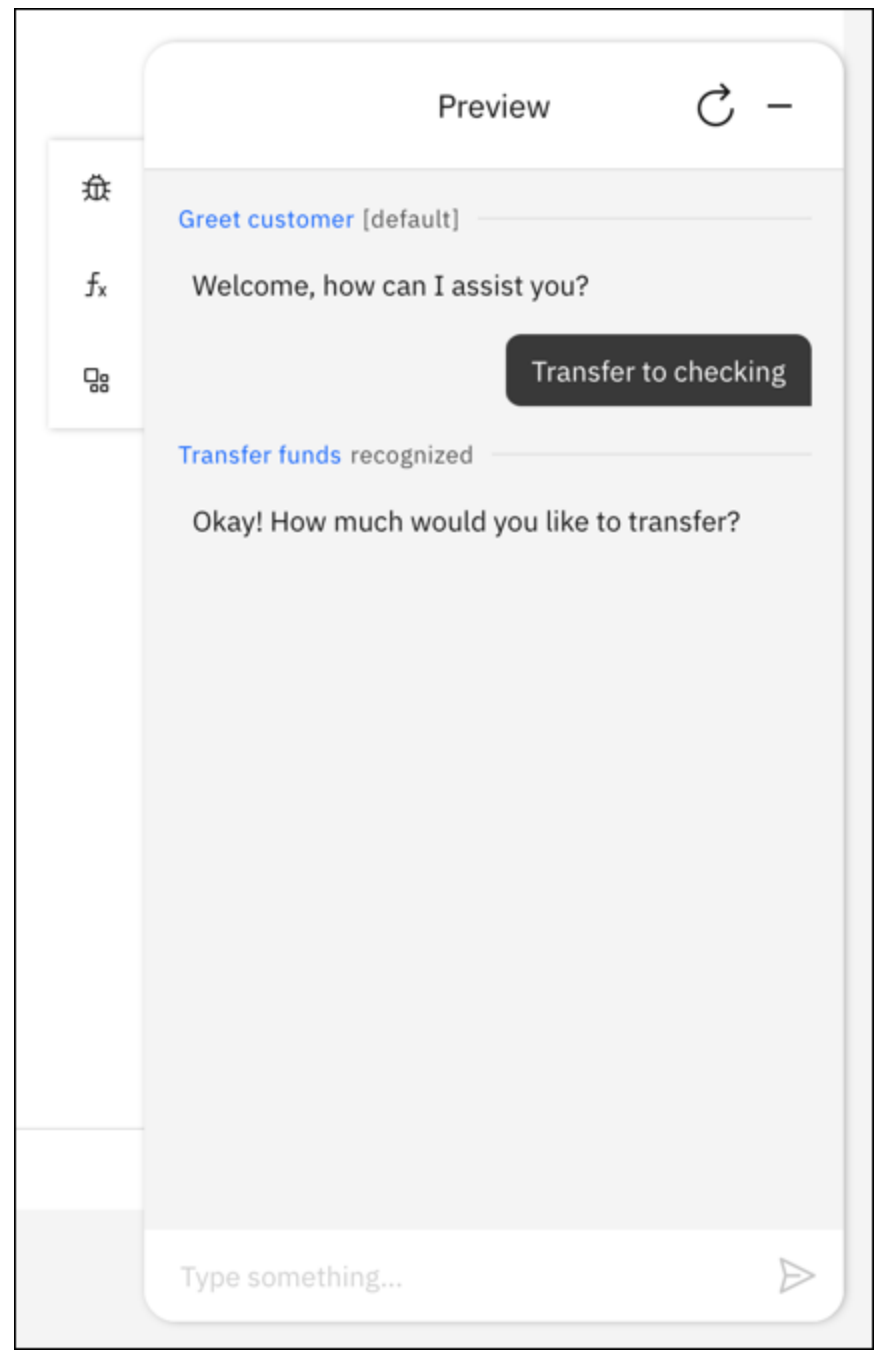
Using Preview to test your action

You can test the action at any time to see whether the resulting interaction works as intended. The **Preview** in the action pages, shows you what customers see when they use web chat to interact with this action.

Note: Before you test your action, make sure you save any new changes, and wait until the system finishes training. If the system is still training, a message is displayed that says so.

1. Click **Preview**. The Greet customer action starts.
2. In the chat window, type some text and then press Enter.
3. Check the response to see if your assistant correctly interpreted the input, started the intended action, and performed the appropriate step.

   The Preview pane names the action that was recognized in the input.



Preview

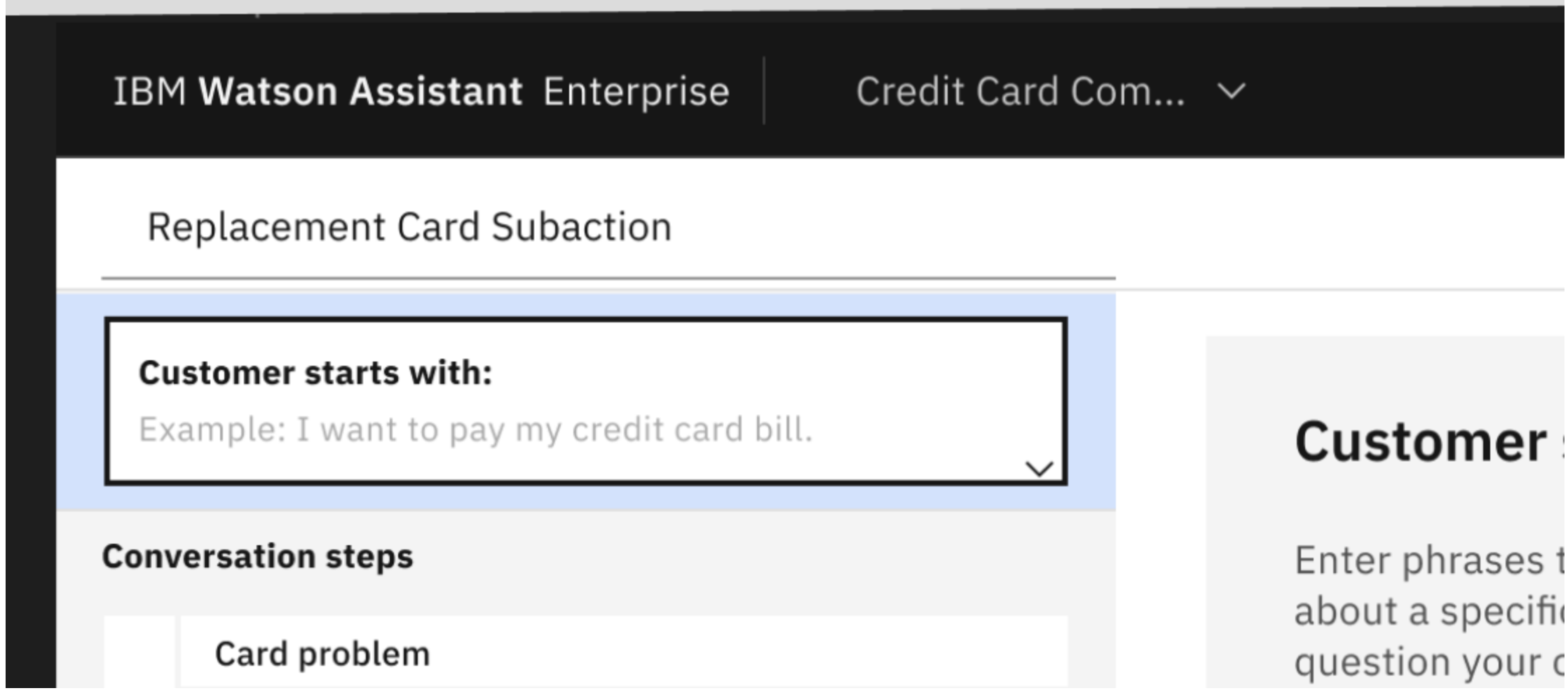
If the assistant doesn't understand a phrase, you see the built-in action `No matches`.

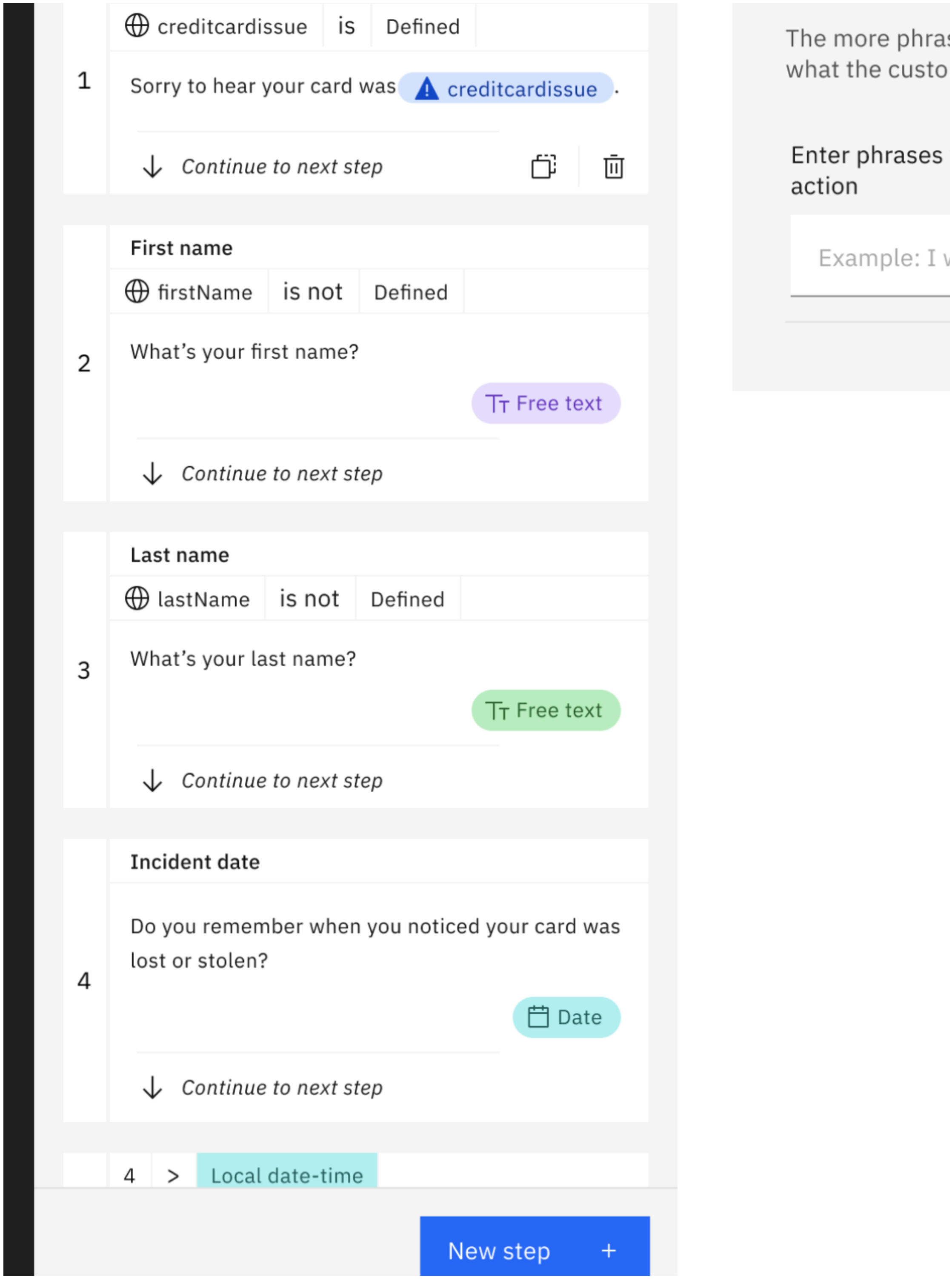
4. Continue to converse with your assistant to see how the conversation flows.
5. To remove prior test utterances from the chat pane and start over, click the **Reset** icon. Not only are the test utterances and responses removed, but this action also clears the values of any variables that were set as a result of your previous interactions.

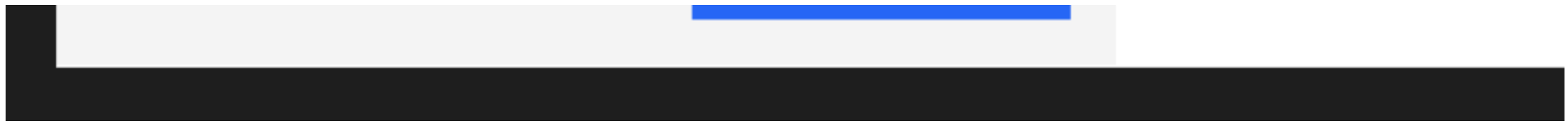
Note: Queries that you submit through the Preview pane generate `/message` API calls, but they are not logged and do not incur charges.

Using Preview to test your action with dialog activated

In the action editor, if dialog is activated, the preview panel shows an error message:




creditcardissue is Defined
1
Sorry to hear your card was creditcardissue .
Continue to next step
First name
firstName is not Defined
2
What's your first name?
Free text
Continue to next step
Last name
lastName is not Defined
3
What's your last name?
Free text
Continue to next step
Incident date
4
Do you remember when you noticed your card was lost or stolen?
Date
Continue to next step
4 > Local date-time
New step +
The more phra
what the custo
Enter phrases
action
Example: I



Click **Don’t remind me** to hide the notification until you turns off and then on dialog.

Saving changes before testing

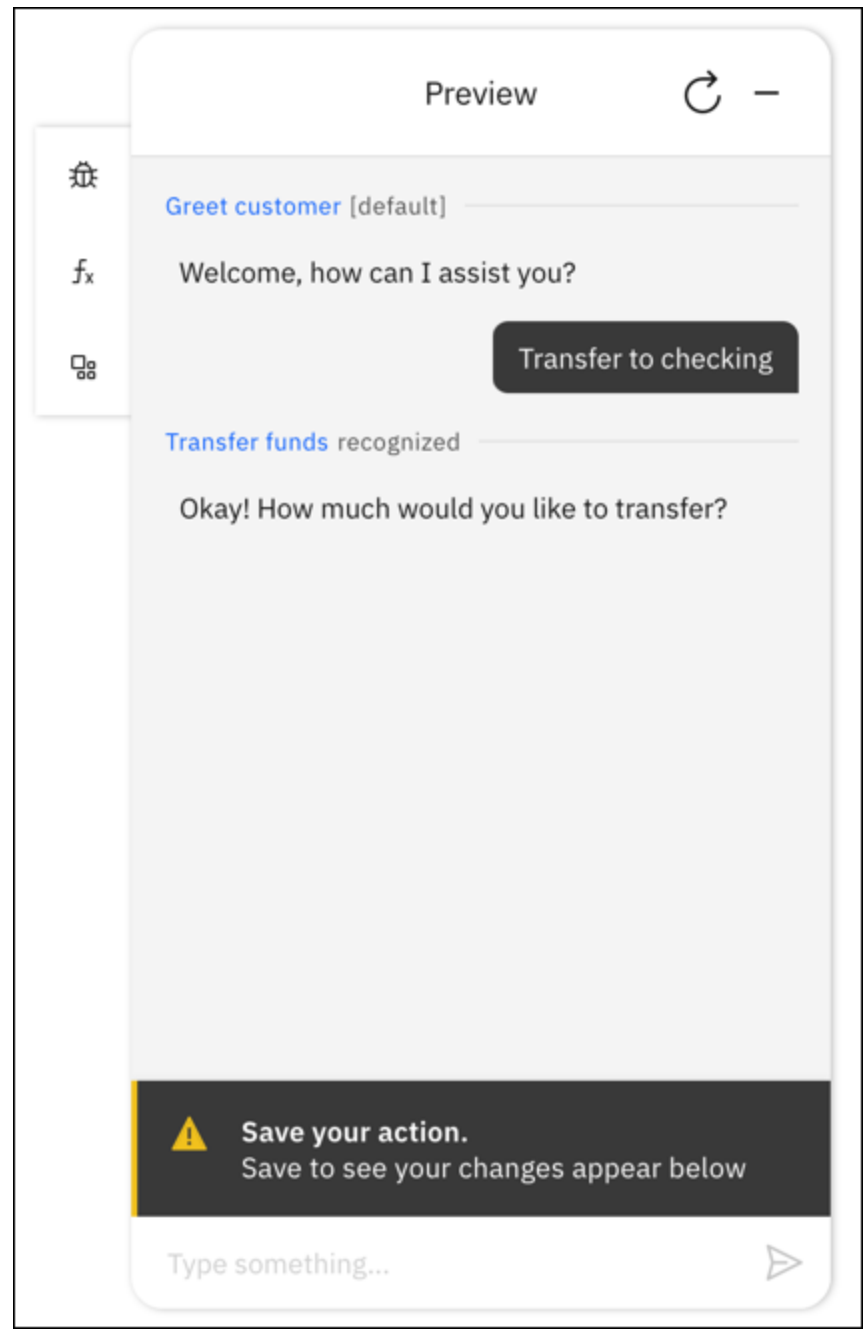
Preview represents updates from the last time that the assistant was saved.

Changes are saved when you:

- Select the save icon
- Click a new step
- Open Preview
- Reset Preview

To learn more about saving changes, see Saving your actions.

If you make several edits without saving, the preview pane shows a message that you need to save before you test your changes.



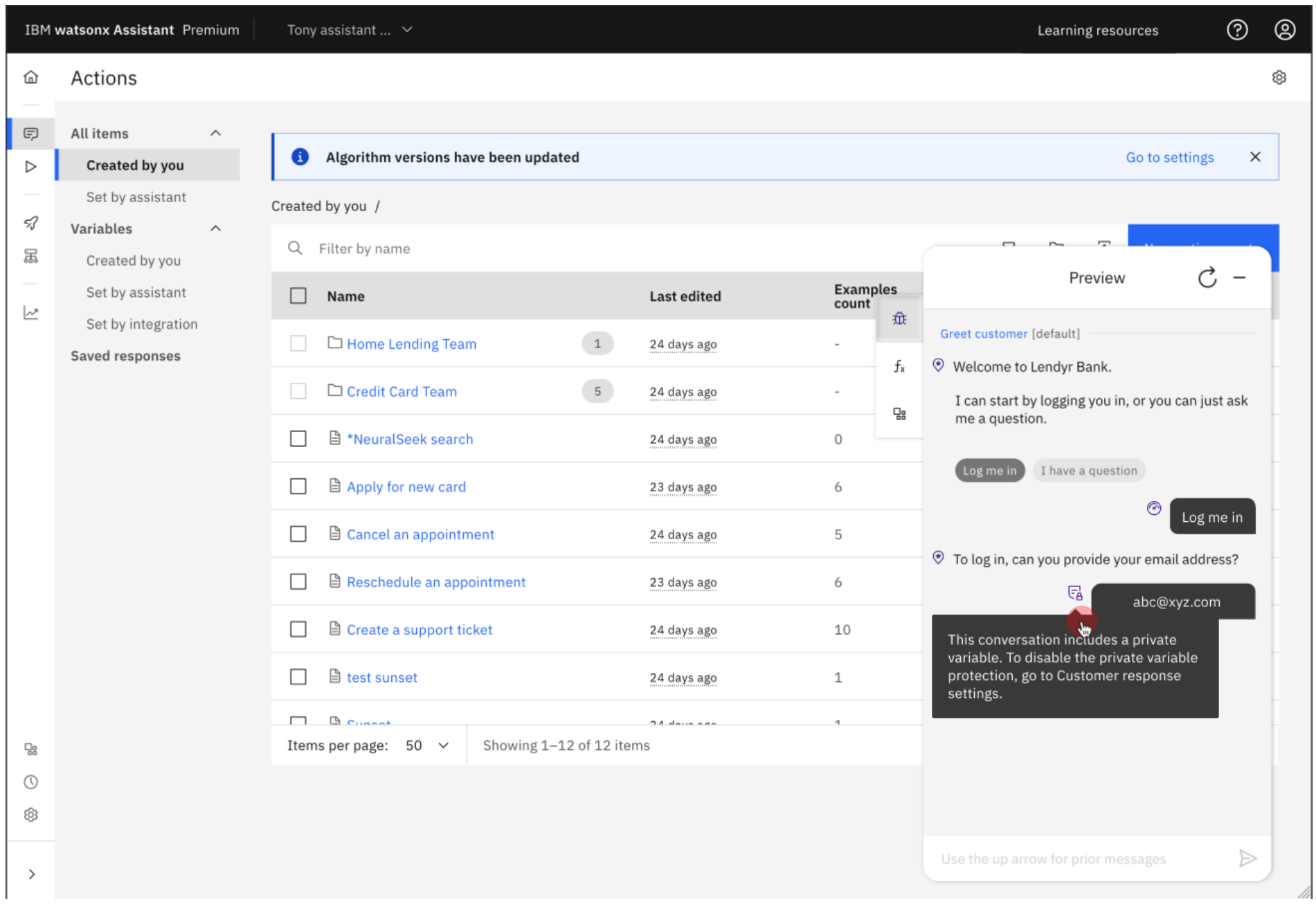
Save your action

Using debug mode in Preview

The **Preview** menu has **Debug mode** that you can turn on to see information related to why the assistant responds or doesn't respond to a particular input.

In **debug mode**, you can identify a conversation that includes a private variable by using the **toggle-tip** icon ( ).

Tip: To disable the private variable protection, go to **Customer response settings** in the **Editor** tab inside an action step.



Private variable in debug mode

Debug mode has four tools to analyze your action:

- Start and end of an action
- Confidence scores
- Step locator
- Follow along

Start and end of an action

The assistant marks the spots in the conversation when a customer enters an input that fits within an action. The assistant also marks when an action completes, and how it completes.

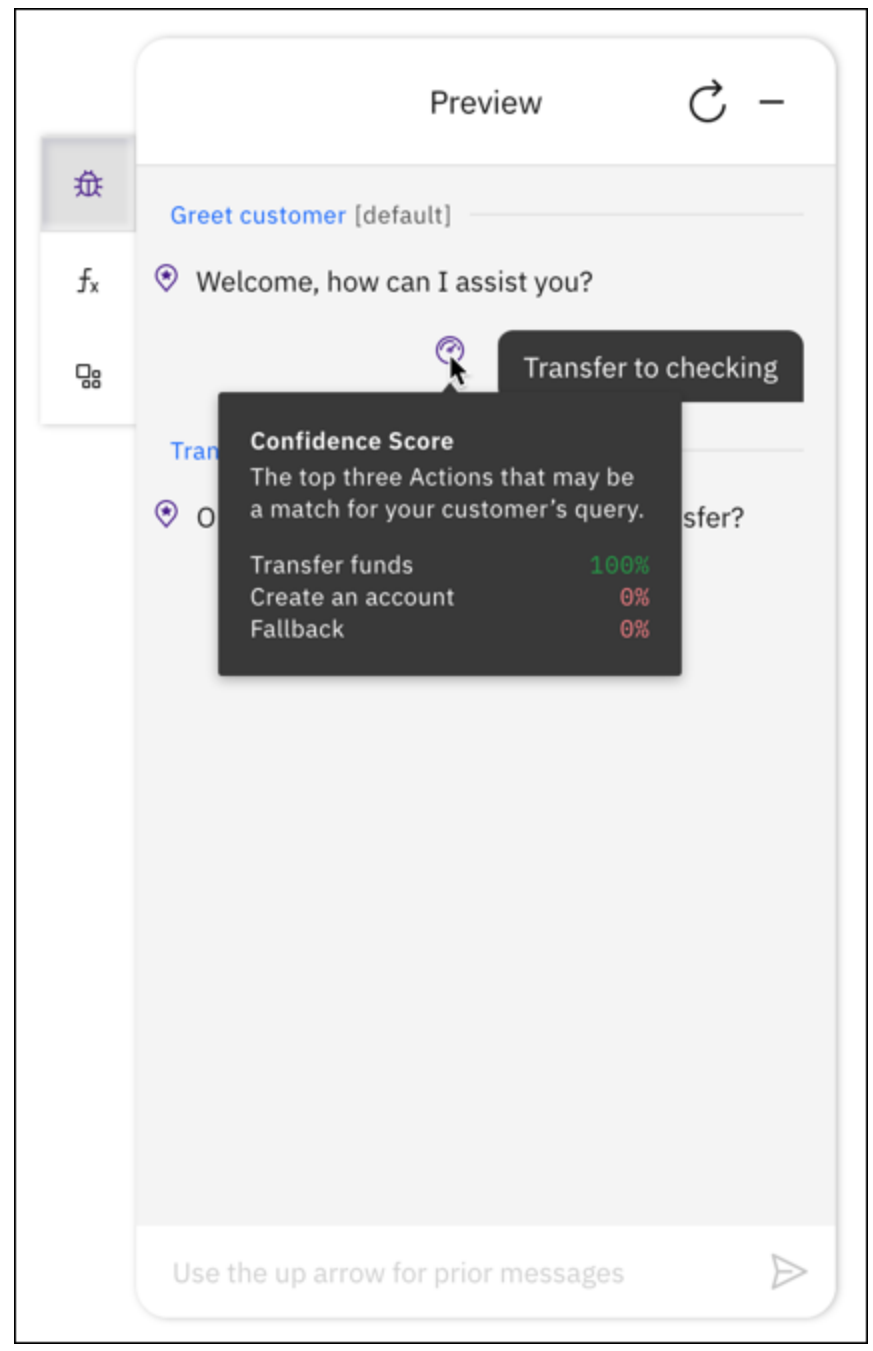
Completion options include ending:

- With an end step
- Without an end step
- With a human agent escalation
- With a search to a knowledge base

Action confidence score

Every input that you enter can start a new topic to show a **confidence score** icon. Hover over the **confidence score** icon to see a list of actions with different confidence scores.

The confidence scores represent the assistant's confidence that the sentence or phrase that you entered can be solved by the steps that are built into a specific action.



Debug mode

The top score in green represents the action with the highest confidence and the one the assistant used.

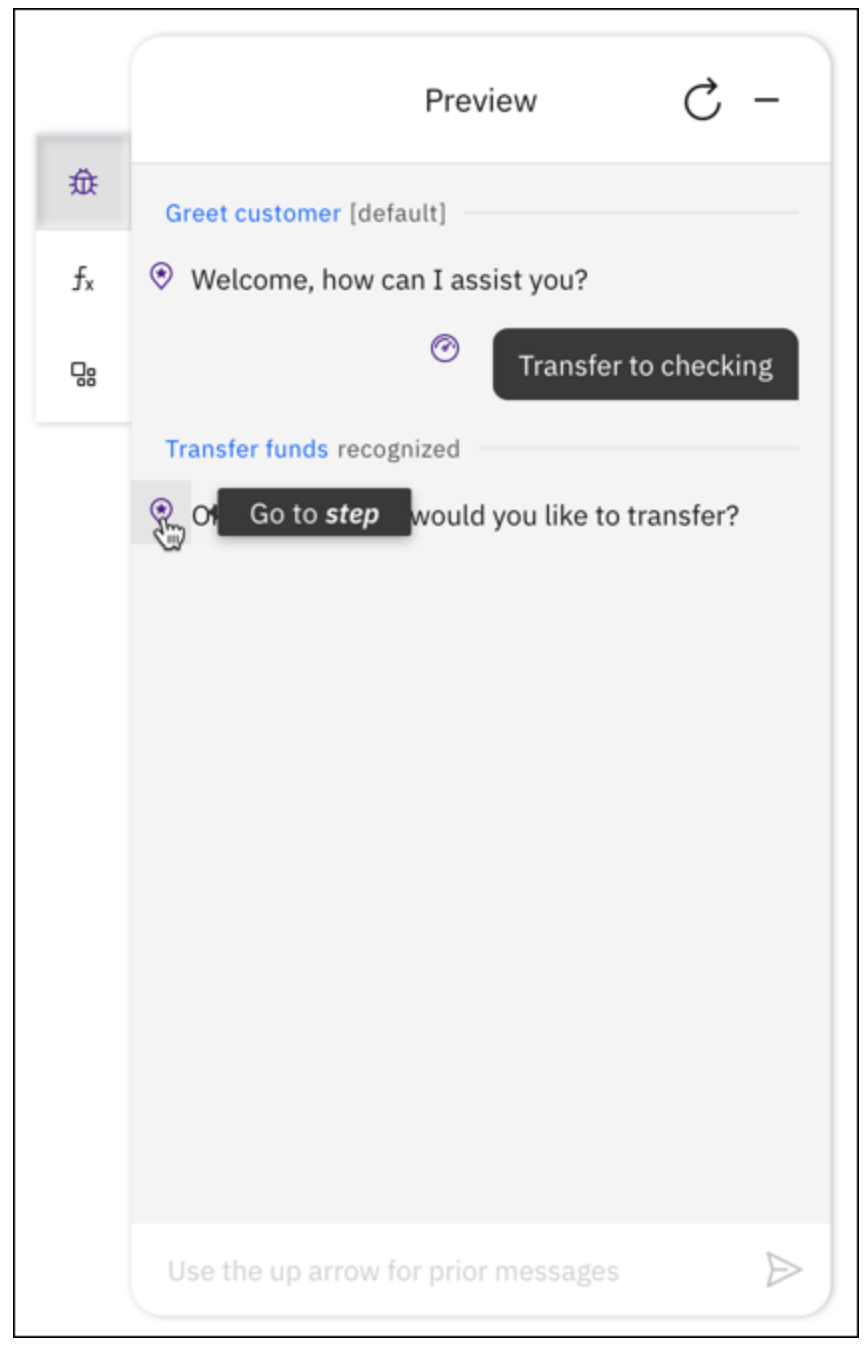
The remaining two are actions that were considered because of their confidence score, but weren't used because the confidence scores were lower.

If no action scores higher than 20% confidence, you see the built-in action `No matches`.

Step locator

Sometimes you might find an error in the middle of a test conversation, and need to find which step and action is involved. A locator icon next to each assistant response lets you find the associated steps in the editor.

Click the icon, and the editor shows the corresponding step in the background.



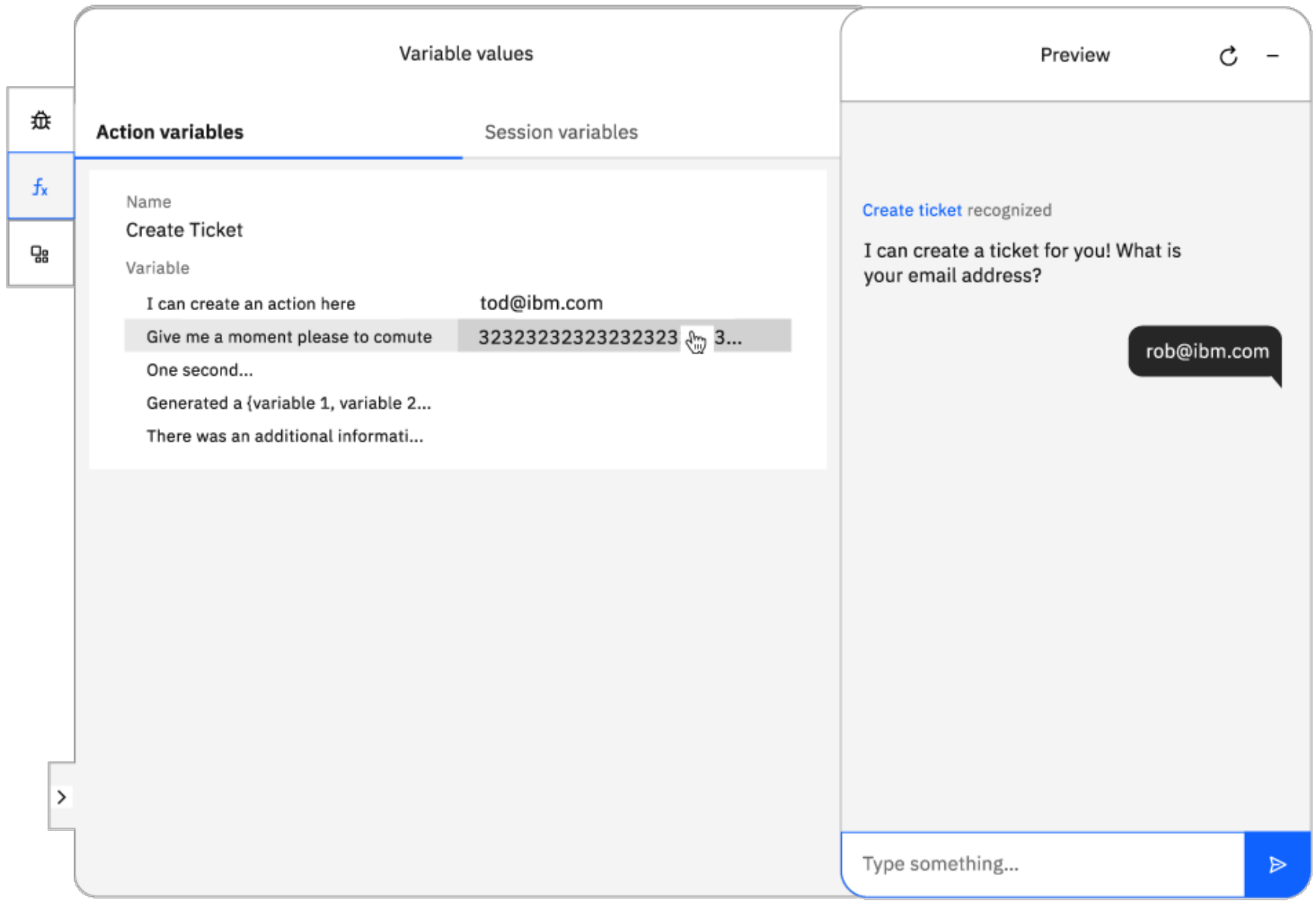
Step locator

Follow along

Follow along connects what you are seeing in Preview with what you built in the action. As you interact with your assistant, the debug mode automatically opens each step in the background. That means you can fix an error as soon as you see it, because the editor is already open to the corresponding step.

Variable values in Preview

In **Preview**, you can test your conversation by verifying the variable values. You click **Variable values** to see the values stored in each variable. The **Variable values** pane has two tabs, one for action variables and another one for session variables.



Variable values

Note: For better visibility of long variable values, you can expand the width of the debug mode panel by using the **Expand** icon.

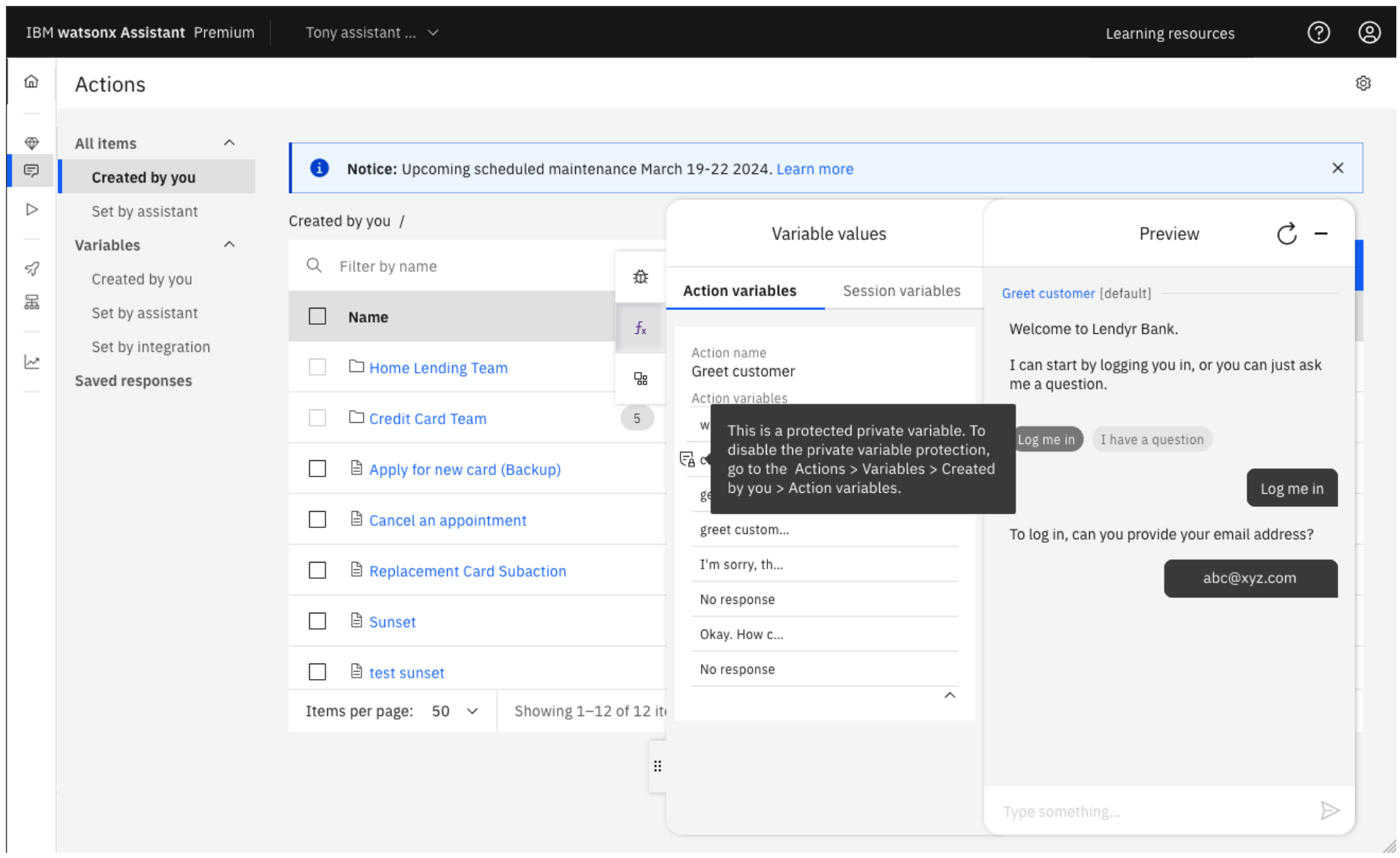
To learn more about variables, see Managing information during the conversation.

The private variables appear as masked texts in the conversation. You can identify a conversation that includes a private variable with the toggle-tip icon ( ) that appears next to the conversation.

Action variables in Preview

In the **Action variables** section, identify a private variable with the toggle-tip icon ( ) that appears next to the conversation.

Tip: To disable the private variable protection, go to **Actions** > **Variables** > **Created by you** > **Action variables**. In **Action variables** clear **Protect data stored in this variable** checkbox.



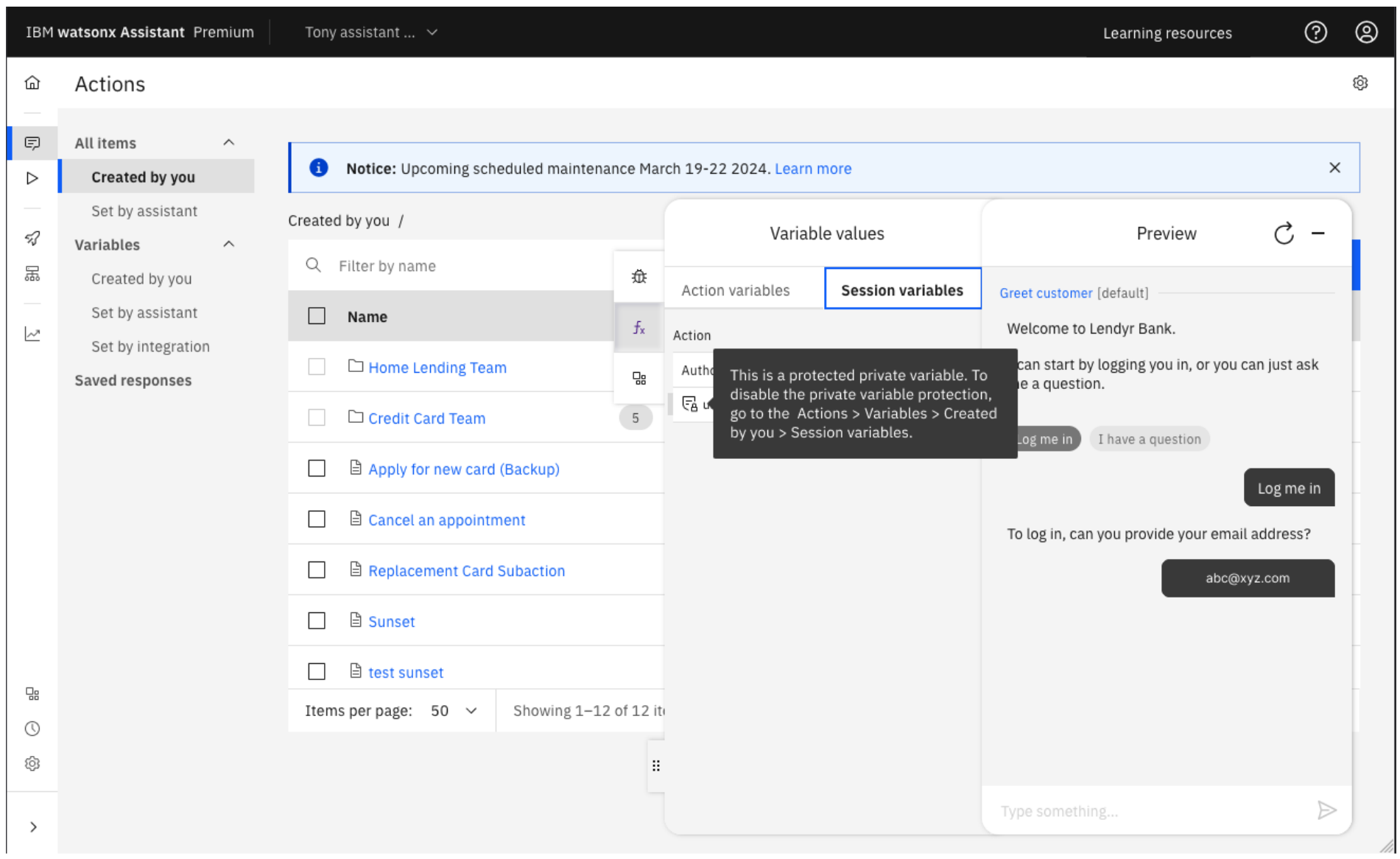
Private variable in action variables

Session variables in Preview

If you are using dialog, you can see session variables for both actions and dialog on the **Session variables** tab.

In the **Session variables** section, you can identify a private variable with the toggle-tip icon ( ) that appears next to the conversation.

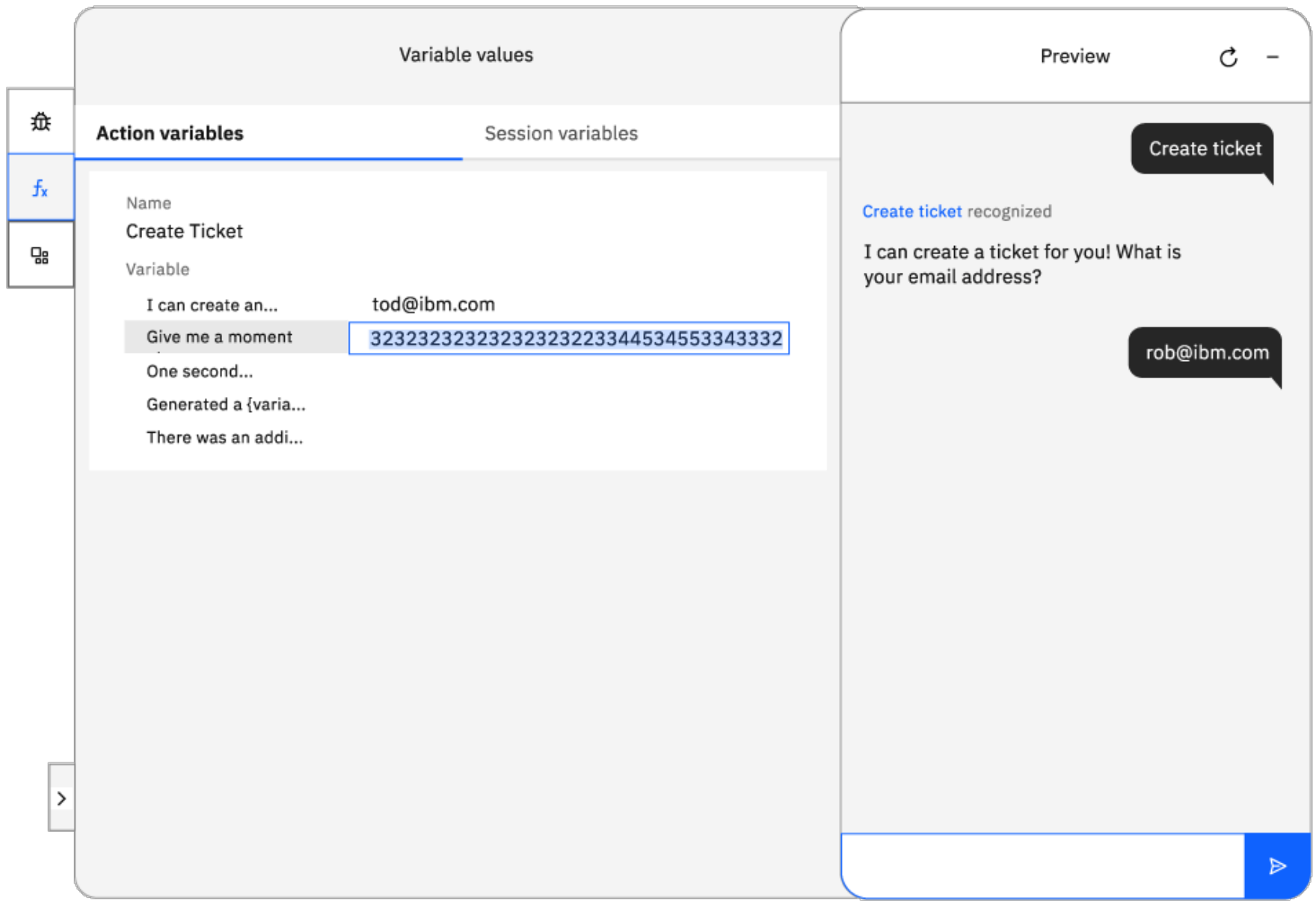
Tip: To disable the private variable protection, go to **Actions** > **Variables** > **Created by you** > **Session variables**. In **Session variables** clear **Protect data stored in this variable** checkbox.



Private variable in session variables

Editing the variable values

In the debug mode, you can edit the variable values by clicking on the value that you want to edit. For better visibility of long variable values, you can resize the width of the debug mode panel by using the **Resize** tab (<). To edit large values for action variables, you can click the **Expand** button next to the variable value to open the **Editor** window. The **Editor** window contains the variable name and the value that is associated with it. You can use the **Editor** window to edit the variable value.

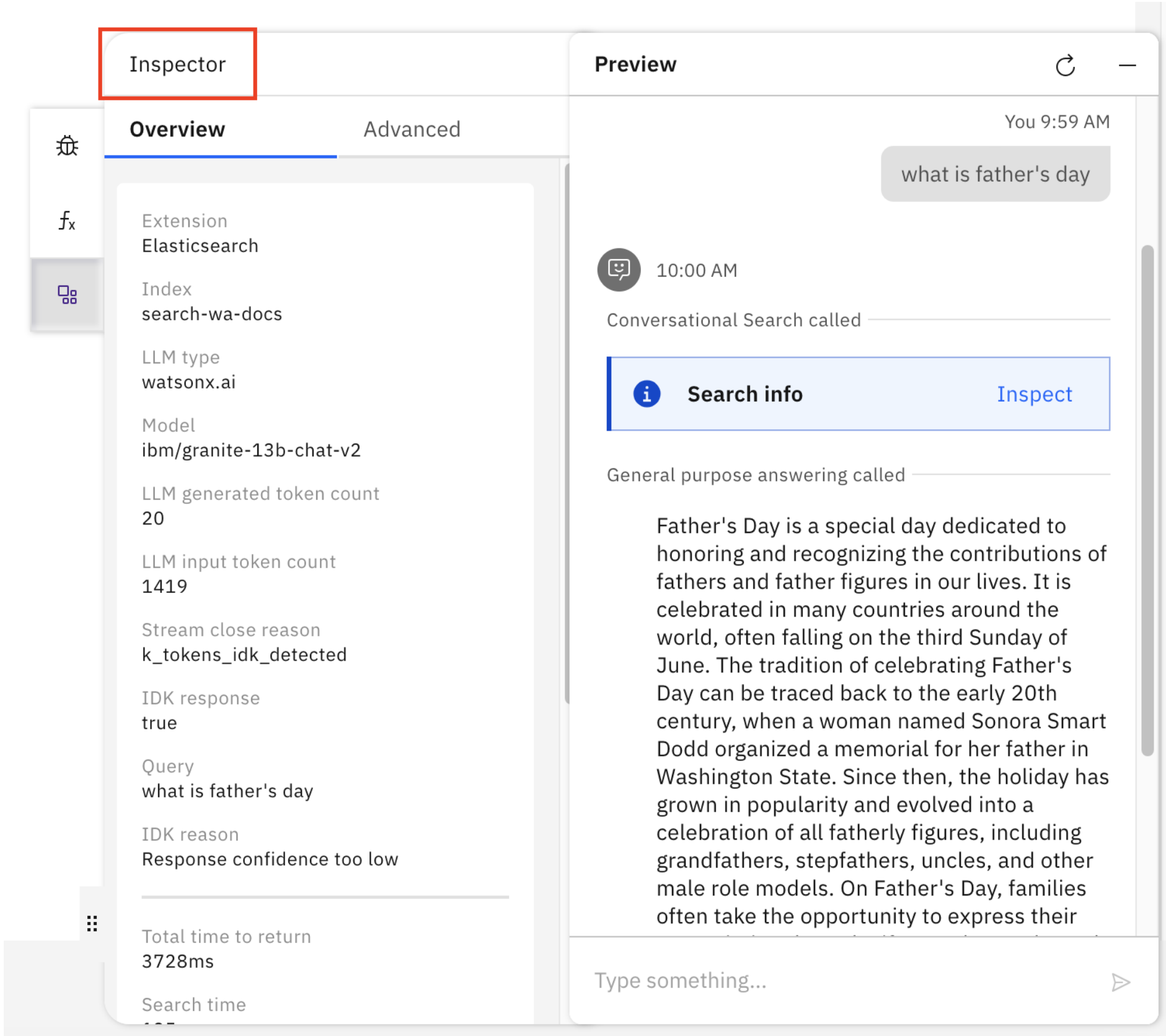


Edit variable values

Using Inspector in Action preview

If you are using a custom extension in your action, you can use the **Inspector** in Preview to debug. For more information, see Debugging custom extensions.

If you are using conversational search or skill-based actions, you can use the **Inspector** in Preview to debug. For more information, see Debugging conversational search.



Inspector

Global settings for actions

Autolearning feature in watsonx Assistant

Effective 16 June 2025, the autolearning feature is withdrawn for watsonx Assistant. After this date, autolearning settings are removed from the **Actions global settings** page, and all autolearning capabilities are disabled.

Use **Global settings** to configure features across all actions.

On the **Actions** page, click **Global settings** .

Global settings provide options, configurations, and tasks for:

- Conversation routing
- Generative AI
- Autocorrection
- Display formats
- Algorithm version
- Upload/Download
- Autolearning

Conversation routing

In the **Conversation routing** tab, you have the following settings:

- Ask clarifying questions

- Customize modes
- Change conversation topic
- No matches

Note: Search routing functionality is moved to Generative AI page of the assistant home screen.

Clarifying questions

On the **Clarifying questions** tab, you can customize how an action asks clarifying questions.

In the **Ask clarifying questions** section, you can:

- Enable or disable if your assistant disambiguates (asks a clarifying question).
- Modify the text that your assistant uses to introduce the clarification list or when no action matches.
- Enable or disable response modes, and modify the text that your assistant uses with a response mode. If you enable response modes, you can use the **Customize modes** section to choose a response mode for each action to set how it behaves.

In the **No matches** section, you can affect how often your assistant routes customers to the **No matches** action when input is unrecognized.

For more information, see:

- Customizing clarifying questions
- Response modes
- When the assistant can't understand your customer's request

Change conversation topic

The **Change conversation topic** feature enables your assistant to handle digressions, dynamically responding to the user by changing the conversation topic as needed. For more information, see Allowing your customers to change the topic of the conversation.

If necessary, you can disable changing the topic for all actions:

1. On the **Change conversation topic** tab, set the switch to **Off**.
2. Click **Save**, and then click **Close**.

Allow change of topic between actions and dialog

If you are using actions and dialog, you can ensure that customers can change topics between an action and a dialog node.

Note: This setting is available if you activate dialog in Assistant settings. For more information, see Activating dialog and migrating skills.

1. Set the toggle **Change topics from actions to dialog** to **On**.
2. Click **Save**, and then click **Close**.

Generative AI

On the **Generative AI** tab, you can enable or disable the generative AI capabilities of the assistant such as *information gathering*.

When you enable the *information gathering* feature, your assistant uses a large language model (LLM) in watsonx.ai to intelligently recognize multiple pieces of information in the customer responses and fill the corresponding steps to avoid multiple prompts in a session.

For more information, see Using watsonx.ai for generative AI capabilities.

Confirmation to return to previous topic

By default, new assistants are set to ask a `yes or no` question to confirm that the customers want to return to the previous action. You can modify these settings in the Confirmation section. For more information, see Confirmation to return to previous topic.

Autocorrection

Autocorrection fixes misspellings that users make in their requests. The corrected words are used to match to an action.

Autocorrection is enabled automatically for all English-language assistants. It is also available in French-language assistants, but is disabled by default. The autocorrection setting isn't available for any other languages.

For more information, see Autocorrecting user input.

Display formats

Use display formats for variables that use date, time, numbers, currency, or percentages. You can also choose a default locale to use if one isn't provided by the client application. You can ensure that the format of a variable in the web chat is what you want for your assistant. For example, you can choose to have the output of a time variable appear in HH:MM format instead of HH:MM:SS.

Variables are formatted by using a system default unless you specify otherwise.

Display format setting	Description	English - United States (en-US) examples
Locale	Choose a default locale for the assistant if one can't be determined. The locale that you choose uses formats specific to that country and language. If you choose a locale, the date, time, number, currency, and percentage format fields change to show choices specific to that locale. The system default is `English - United States (en-US)`.	
Date	For calendar dates, choose short, medium, long, full, YY/MM/DD, or YYYY/MM/DD	• Short: `1/31/23` • Medium: `Jan 31, 2023` • Long: `January 31, 2023` • Full: `Tuesday, January 31, 2023`
Time	For times, choose short or medium	• Short: `1:53 PM` • Medium: `1:53:30 PM`
Number fraction digits	Use the system default (up to 14 digits) or two digits	`10.12`
Number delimiter	Use the system default, none, comma, or period	• None: `2000.12` • Comma: `2,000.12` • Period: `2.000,12`
Currency fraction digits	Use the system default (up to 14 digits) or two digits	`10.99`
Currency symbol	Use the system default or choose a global symbol	`$10.99`

Percentage fraction digits	Use the system default (up to 14 digits) or two digits	10.75%

Display format settings

Algorithm version

Choose which watsonx Assistant algorithm to apply to your future trainings. For more information, see Algorithm version.

Upload/Download

You can upload or download actions.

Downloading

To back up actions, download a JSON file and store it. On the **Upload/Download** tab, click the **Download** button.

Uploading

To reinstate a backup copy of actions that you exported from another service instance or environment, import the JSON file of the actions you exported.

⚠ **Important:** If the watsonx Assistant service changes between the time you export the actions and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before.

On the **Upload/Download** tab, drag a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

⚠ **Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

Response modes Beta

IBM Cloud

You can choose a response mode for each action to set how it behaves. The modes are *clarifying* and *confident*.

Clarifying mode: Start here. In the clarifying mode, your assistant is eager to ask questions so you can ensure that your customer gets to the action they need. An assistant is more likely to ask questions to be sure an action matches what a customer is asking. A new or untested action gets the training that it needs.

Confident mode: Take the next step. After you use analytics to improve your assistant, use the confident mode. Your assistant solves customer issues with authority and accuracy. An assistant is less likely to ask questions and is more likely to trigger actions that match. Use confident mode after you test and train actions.

Response modes are a beta feature that is available for evaluation and testing purposes on IBM Cloud only.

Settings

Settings for the two modes are in the global settings. For more information, see Global settings for actions.

You can choose what mode to use when you create a new action. Clarifying mode is the default and is designed for use with new, untested actions that need training.

The settings are:

Clarify when one action matches: If an assistant prioritizes one action that it thinks matches the customer's request, it can clarify the match by asking the customer to confirm. This clarification helps you ensure that the action is the right one and allows the customer to give input before proceeding. For example, if the assistant thinks that a single action named `Pay Bill` is the right one, it can clarify the choice by asking the customer `Did you mean: Pay Bill`.

Clarify when more than one action matches : When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. Instead of guessing which action to use, your assistant shows a list of the possible actions to the customer and asks the customer to pick the right one.

Offer support option when asking a clarifying question : The assistant can include a choice to connect to other support. If the customer picks this choice, the assistant uses your Fallback action.

Step validation attempts before offering support : If a customer provides invalid answers for a step in an action, the assistant can offer to connect to other support in the Fallback action. The step validation count measures how many invalid answers can occur before the assistant provides this choice.

This table shows the default settings for each mode.

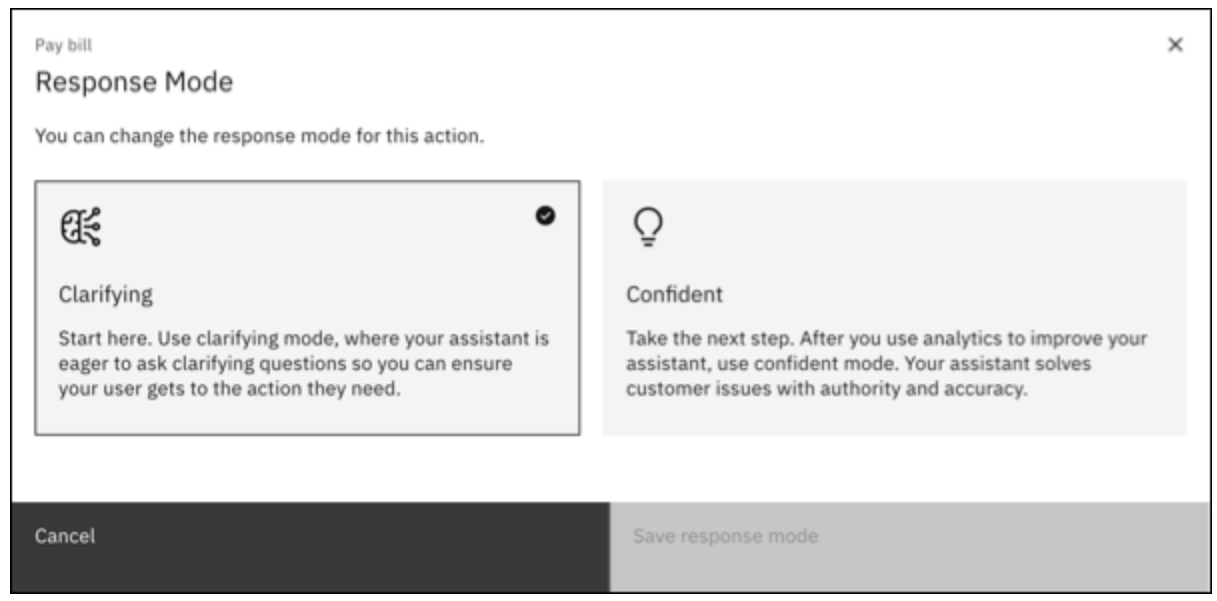
	Clarifying	Confident
Clarify when one action matches	More often	Sometimes
Clarify when more than one action matches	More often	Sometimes
Offer support option when asking a clarifying question	More often	Sometimes
Step validation attempts before offering support	2 times	4 times

Default settings

Choosing a mode for individual actions

When you edit an action, you can see the mode that it uses and change it if you need to.

1. Click the Action response mode icon . The mode in use is checked.



Action response mode

2. Click the other mode if you want to change it, and then click **Save response mode**.

Override system defaults

Enterprise

For Lite and Plus plans, the default settings can't be changed. For Enterprise plans, you can override system defaults to customize each mode.

To override system defaults:

1. On the **Actions** page, click **Global settings** .
2. On the **Clarifying questions** tab, click **Customize modes**.
3. Set the **Override System Defaults** toggle to **On**. This toggle is available only for Enterprise plans.
4. Use the menu to modify any of the settings for either mode.

If you customize, the choices for each setting are;

Setting	Choices
Clarify when one action matches	More often, Sometimes, Less often
Clarify when more than one action matches	More often, Sometimes, Less often
Offer support option when asking a clarifying question	More often, Sometimes, Less often

Step validation attempts before offering support	2 times, 3 times, 4 times

Mode customization choices

Note: If you customize modes, be sure to test any changes.

Autocorrecting user input

Autocorrection fixes misspellings that users make in their requests. Corrected words are used to match to an action or an intent.

Autocorrection corrects user input in the following way:

- Original input: `letme applt for a memberdhip`
- Corrected input: `let me apply for a membership`

When your assistant evaluates whether to correct the spelling of a word, it does not rely on a simple dictionary lookup process. Instead, it uses a combination of natural language processing and probabilistic models to assess whether a term is, in fact, misspelled and should be corrected.

By default, autocorrection is enabled in all assistants that use English. However, it is disabled by default in all assistants that use French. You can enable or disable **Autocorrection** by going to **Global settings** > **Autocorrection**.

Important: Autocorrection is not available for search integration in your assistant and for assistant languages other than English and French.

Disabling autocorrection

If necessary, you can disable autocorrection for your assistant.

Note: If you find that a domain-specific term is being corrected that shouldn't be, you can prevent the correction from happening by adding the term or phrase to your training data. For more information, see Autocorrection rules.

If you are using actions in your assistant, follow these steps to disable autocorrection:

1. On the **Actions** page, click **Global settings**.
2. Click the **Autocorrection** tab.
3. Set the switch to **Off**, then click **Save**.

If you are using dialog in your assistant, follow these steps to disable autocorrection:

1. In the **Options** section, click **Autocorrection**.
2. Set the switch to **Off**.

Testing autocorrection in dialog

If you are using dialog, you can test autocorrection by using **Try it out**.

1. In **Try it out**, enter a request that includes some misspelled words.

   If words in your input are misspelled, they are corrected automatically, and an icon is displayed. The corrected utterance is underlined.

2. Hover over the underlined utterance to see the original wording.

If there are misspelled terms that you expected your assistant to correct, but it did not, then review the rules that your assistant uses to decide whether to correct a word to see if the word falls into the category of words that your assistant intentionally does not change.

Autocorrection rules

To avoid overcorrection, your assistant does not correct the spelling of the following types of input:

- Capitalized words
- Words with an uppercase character
- Emojis

- Locations, such as states and street addresses
- Numbers and units of measurement or time
- Proper nouns, such as common given names or company names
- Text within quotation marks
- Words containing special characters, such as hyphens (-), asterisks (*), ampersands (&), or at signs (@), including those used in email addresses or URLs.
- Words that *belong*, meaning words that have implied significance because they occur in your action steps or dialog entity values, entity synonyms, or intent user examples.

How is spelling autocorrection related to fuzzy matching?

In dialog, *fuzzy matching* helps your assistant recognize dictionary-based entity mentions in user input. It uses a dictionary lookup approach to match a word from the user input to an existing entity value or synonym in the skill's training data. For example, if the user enters `boook`, and your training data contains a `@reading_material` entity with a `book` value, then fuzzy matching recognizes that the two terms (`boook` and `book`) mean the same thing.

In dialog, when you enable both autocorrection and fuzzy matching, the fuzzy matching function runs before autocorrection is triggered. If it finds a term that it can match to an existing dictionary entity value or synonym, it adds the term to the list of words that *belong* to the skill, and does not correct it.

For example, if a user enters a sentence like `I wnt to buy a boook`, fuzzy matching recognizes that the term `boook` means the same thing as your entity value `book`, and adds it to the protected words list. Your assistant corrects the input to be, `I want to buy a boook`. Notice that it corrects `wnt` but does *not* correct the spelling of `boook`. If you see this type of result when you are testing your dialog, you might think your assistant is misbehaving. However, your assistant is not. Thanks to fuzzy matching, it correctly identifies `boook` as a `@reading_material` entity mention. And thanks to autocorrection revising the term to `want`, your assistant is able to map the input to your `#buy_something` intent. Each feature does its part to help your assistant understand the meaning of the user input.

Algorithm version and training

You can use the **Algorithm version** setting to choose which watsonx Assistant algorithm is used for training. Your assistant is trained when you update the content or settings, or through automatic retraining.

There are three choices:

- **Beta**: Use Beta to preview and test what is coming. The capability in the beta version is likely to become a supported version later on. It's not recommended to use the beta version in a production deployment.
- **Latest**: The current supported version that's recommended for your live production assistant.
- **Previous**: The last supported before the latest version. Support for this version ends when the next version is released.

To choose an algorithm version for actions:

1. On the **Actions** page, click **Global settings** ⚙.
2. Click the **Algorithm Version** tab.
3. Choose a version, then click **Save**.

To choose an algorithm version for dialog:

1. On the **Dialog** page, open the **Options** section.
2. Click **Algorithm Version**.
3. Choose a version.

The latest and previous versions have date labels such as **Latest (15-Apr-2023)** or **Previous (20-Dec-2022)**. See the watsonx Assistant release notes for details about each algorithm version release.

Algorithm version choices are currently available for Arabic, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, English, French, German, Japanese, Korean, Italian, Portuguese, and Spanish. The universal language model uses a default algorithm.

Automatic retraining

IBM Cloud

IBM® watsonx™ Assistant was released as a service in July 2016. Since then, users have been creating and updating skills to meet their virtual assistant needs. Behind the scenes, watsonx Assistant creates machine learning (ML) models to perform various tasks on the user's behalf.

The primary ML models deal with action recognition, intent classification, and entity detection. For example, the model might detect what a customer

intends when they say `I want to open a checking account`, and what type of account the customer is talking about.

These ML models rely on a sophisticated infrastructure. There are many intricate components that are responsible for analyzing what the user says, breaking down the user's input, and processing it so the ML model can more easily predict what the user is asking.

Since watsonx Assistant was first released, the product team has made continuous updates to the algorithms that generate these sophisticated ML models. Older models continued to function while running in the context of newer algorithms. Historically, the behavior of these existing ML models did not change unless the skill was updated, at which point the skill was retrained and a new model that is generated to replace the older one. This meant that many older models never benefited from improvements in our ML algorithms.

IBM® watsonx™ Assistant uses continuous retraining. The watsonx Assistant service continually monitors all ML models, and automatically retrains those models that have not been retrained in the previous 6 months. Your assistant retrains by using the selected algorithm version. If the version you selected is no longer supported, watsonx Assistant retrains by using the version that is labeled as **Previous**. This means that your assistant automatically has the supported technologies that are applied. Assistants that were modified during the previous 6 months are not affected.

Note: Usually, this retraining is seamless from a customer point of view. The same inputs result in the same actions, intents, and entities being detected. Sometimes, the retraining might cause changes in accuracy.

Instructions for watsonx Assistant for IBM On-premises

IBM Cloud Pak for Data | IBM Software Hub

When you upgrade your instance of watsonx Assistant for IBM Cloud Pak for Data, if your existing models were trained by using an algorithm version that is still supported, your models are not retrained during or after the upgrade.

New algorithm versions are included in major releases (for example, 4.0.0 or 5.0.0) or minor releases (for example, 4.5.0 or 4.6.0). A monthly release might include a new algorithm version if there is more than 6 months between major or minor releases. Each algorithm version supports 2 major or minor releases or a maximum time of 12 months, whichever is first. For more information, see the IBM Cloud Pak for Data Software Support Lifecycle Addendum.

Each new release includes full support for the version that is listed as **Latest** in the most recent prior release. This version is then labeled as **Previous** after you upgrade. In addition, each new release supports running models that were trained on the version before that so that upgrading doesn't affect your runtime. For example, if you upgrade from IBM Cloud Pak for Data 4.6 to 4.7, and were using **Latest (01-Jun-2022)** that version becomes listed as **Previous (01 June 2022)** and remains your selected version.

Automatic retraining after you upgrade

After your watsonx Assistant for IBM Cloud Pak for Data upgrade is complete, watsonx Assistant performs automatic retraining for any assistant models that were trained by using a version that is no longer supported. In this case, watsonx Assistant automatically retrains your assistant to the **Latest** version. This automatic retraining is required to ensure your ability to run your trained models in your next upgrade.

Best practices

It's recommended to use the **Latest** version in your production deployment of watsonx Assistant for IBM Cloud Pak for Data. This is the default for new assistants. During an upgrade, your settings don't automatically switch existing assistants to use the latest version. If before your upgrade you selected **Latest**, your settings continue to use that version, now labeled as **Previous**. After you upgrade, it's recommended you choose **Latest** and run basic regression tests.

IBM performs robust testing on various data sets to minimize impacts on existing assistants. But given the nature of machine learning models and the nuance and subtlety of natural language processing, you might find some discrepancies from version to version. If you find a major issue through your tests, you can switch your settings and use **Previous** to return to the previous behavior. In this event, we recommend you contact IBM and provide details of your test so that that IBM can support you in the steps to resolve the problem.

It's also recommended that you try the **Beta** version in one of your test systems after you upgrade. This gives you early visibility to changes that are likely to be delivered in a future version, and reduces the probability of negative impacts to your production systems. IBM values both positive and negative feedback from customers who use Beta. You will have the opportunity to shape how the algorithms function before the version is promoted to **Latest** in a future version. If you choose **Beta**, your assistant always trains on the most current beta version.

Using autolearning to improve assistant responses

IBM Cloud

Deprecated: Autolearning feature in watsonx Assistant
Effective June 16, 2025, the autolearning feature is discontinued for watsonx Assistant. After this date, autolearning settings are removed from the **Actions global settings** page, and all autolearning capabilities are disabled.

Use *autolearning* to enable your assistant to learn from interactions with your customers and improve responses.

When customers interact with your assistant, they often make choices. Your assistant can learn from these user decisions.

For example, a customer might ask a question that the assistant isn't sure it understands. The assistant asks a clarifying question so the customer can choose the right action from a list. If customers most often click the same action (option #2, for example), your assistant can learn that option #2 is the best answer.

Next time, the assistant can list option #2 as the first choice, so customers can get to it more quickly. If the pattern persists over time, it can change its behavior even more. Your assistant can return option #2 immediately, rather than asking a clarifying question.

As your assistant learns over time, your customers get the best answer more often and in fewer clicks.

How autolearning works

Before your assistant can learn from customer behavior, it must observe a significant amount of real conversation data. The conversations take place in a channel such as the web chat, or in a custom application.

Logs of conversations and user decisions from your live environment are the data source for observation. Your assistant analyzes the logs to gain insights. (It doesn't watch real-time clicks during a conversation.)

When the assistant observes enough real conversation data from the live environment, it gains insights to help improve your assistant, providing a better customer experience.

When you publish your assistant to the live environment, autolearning starts training the assistant.

Applying autolearning

You can apply autolearning when the following conditions are met:

- *Ask clarifying questions* is enabled in global settings. For more information, see Asking clarifying questions.
- You publish a version of your assistant to the live environment. For more information, see Publishing your content.
- Your customers are interacting with a channel or custom application that is connected to the live environment.

To apply autolearning improvements to your assistant responses:

1. On the **Actions** page, click **Global settings** ⚙.
2. Click the **Autolearning** tab.
3. Ensure the **Use autolearning to modify responses with training from live environment** switch is set to **On**.
4. In the draft environment, you can preview your actions or your assistant. With autolearning set to **On**, preview in the draft environment uses the autolearning training from the live environment. For more information, see Using Preview to test your action or Previewing your assistant.
5. If you are happy with the results from testing the autolearning improvements in the draft environment, publish a new version of your assistant to the live environment to apply the improvements. For more information, see Publishing your content.

Learning from your data

Observations are made of only your customers' choices to improve only your assistant. These observations are not reused by IBM or shared in any way.

This observed user choices data is separate from the log data for which metrics are displayed on the Analyze page. The observation data is also separate from the information that is collected in all but Enterprise plan service instances and used by IBM for general service improvements. You can opt out of such use by specifying an opt-out header in your `/message` API requests. For more information, see Opting out of log data use.

To prevent your own assistant from applying what it learns by observing user choices to your assistant, disable autolearning:

1. In **Global settings**, click the **Autolearning** tab.
2. Set the **Use autolearning to modify responses with training from live environment** switch to **Off**. This immediately disables the modifying of responses in the draft environment.
3. Publish a new version of your assistant to the live environment to completely disable the modifying of responses by autolearning.

Previewing and publishing

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Previewing and sharing your assistant

Internal review is a necessary step in any virtual assistant workflow. You need an environment without customer interactions so your team can test your assistant. The draft environment closely resembles the final experience that your users encounter so you can ensure that you are publishing the optimal end product.

Saving and editing your work in the draft environment

The draft environment contains all your in-progress work in the **Actions**, **Preview**, and **Publish** pages. Use the **Draft environment** tab to manage the draft environment, including adding draft environment integrations (channels and extensions) that you can use for internal testing before deployment. These integrations are unique to the draft environment, and changes to draft integrations don't affect the live environment.

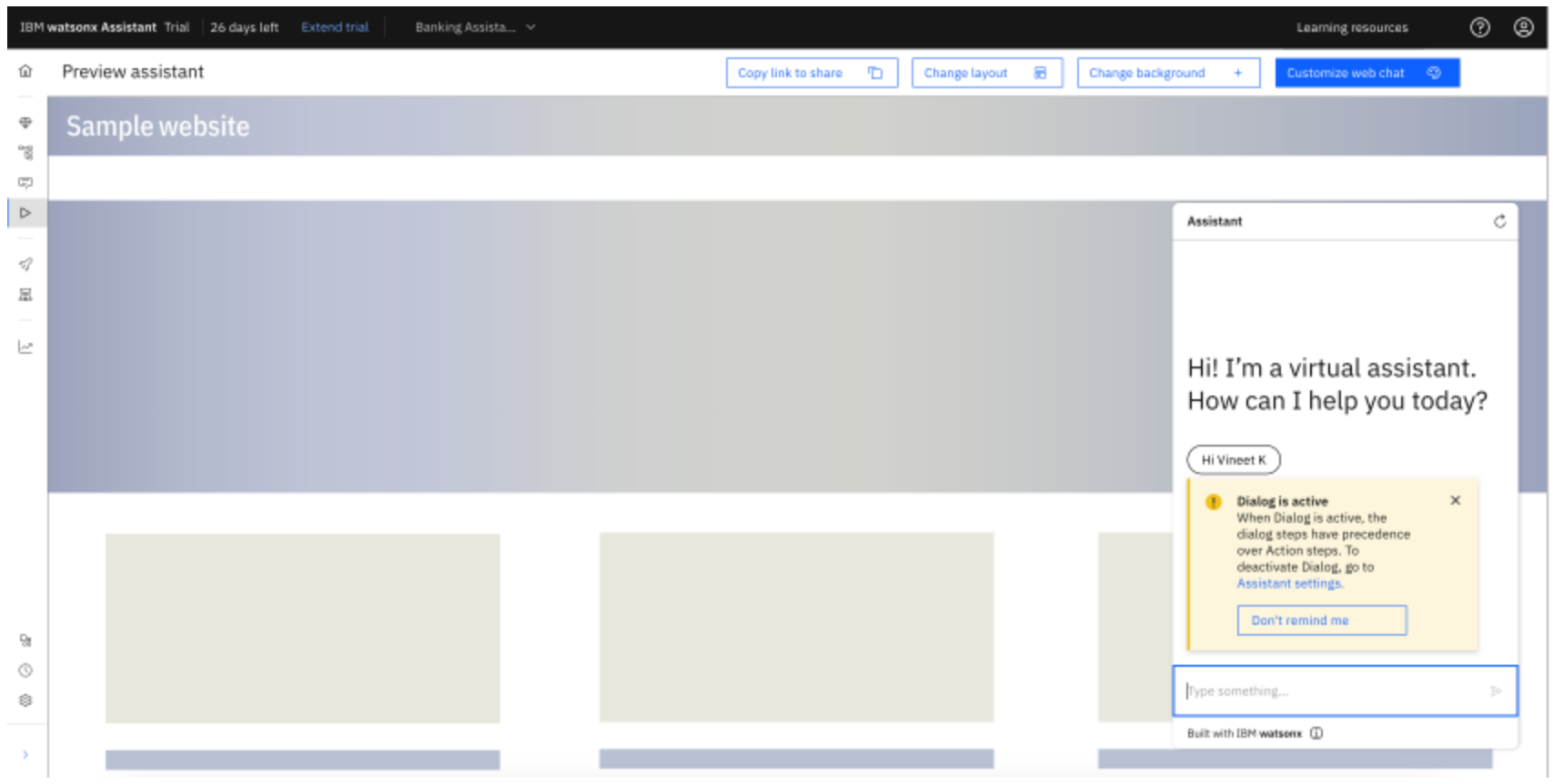
The Preview page

Use the **Preview** page to test your assistant. You can experience your assistant from your customers' perspective. The **Preview** page includes an interactive web chat widget where you can test out your assistant as if you were a customer. The content that is contained in the assistant is the content that you built into your actions or set up with the search integration.

On the **Preview** page, you also find the following elements:

- **Copy link to share**: Share an unauthenticated version of your assistant with your colleagues by sending them a link. For more information, see Copying a link to share.

- **Change background**: Change the background of the page so you can see what your assistant looks like on different web pages. For more information, see Changing background website.
- **Customize web chat**: Customize your draft web chat channel to match your brand or website. For more information, see Web chat setup overview.



Copying a link to share

You can share an unauthenticated version of your assistant with your team by sending them a link. The link opens a sample web page with an interactive web chat widget where you can test out your assistant as if you were a customer. Your subject-matter experts can test your in-progress assistant without needing access to watsonx Assistant itself. The experience is identical to using **Preview this environment** on the draft environment tab.

To share a link:

1. On the **Preview** page, click **Copy link to share**.
2. Send the link to your team.

Note: The preview link is not accessible if web chat security is enabled. For more information about web chat security, see Securing the web chat.

Changing background website

You can visualize how your assistant would look as a web chat widget on your organization's website. You can enter a URL or upload an image.

Entering a URL

You can enter a URL of your organization's website. Your assistant captures an image of your website to use as the **Preview** page background.

Note: Your website must be publicly available to all users. Private or intranet sites can't be accessed. Any login, splash, cookie, or warning screens might be captured in the image.

To enter a URL:

1. On the **Preview** page, click **Change background**.
2. Click **Enter URL**, then click **Continue**.
3. Enter the path of your website URL, for example, `https://www.example.com` or `example.com`.
4. Click **Continue**.

Uploading an image

You can upload an image of your organization's website. Images are stored for 24 hours. The maximum file size is 1 MB. Supported file types are JPEG and PNG.

To upload an image:

1. On the **Preview** page, click **Change background**.
2. Click **Upload an image**, then click **Continue**.

3. Drag a file or click to upload, then click **Change background**.

Images are stored for 24 hours. A warning message might appear on the Preview page about the time limit expiration. To clear this message:

1. On the **Preview** page, click **Change background**.
2. Click **Clear background setting**, then click **Continue**.
3. Click **Remove background** to finish.

Removing the background

After you enter a URL or upload an image to use as a background, you might decide to remove the background and restore the default background.

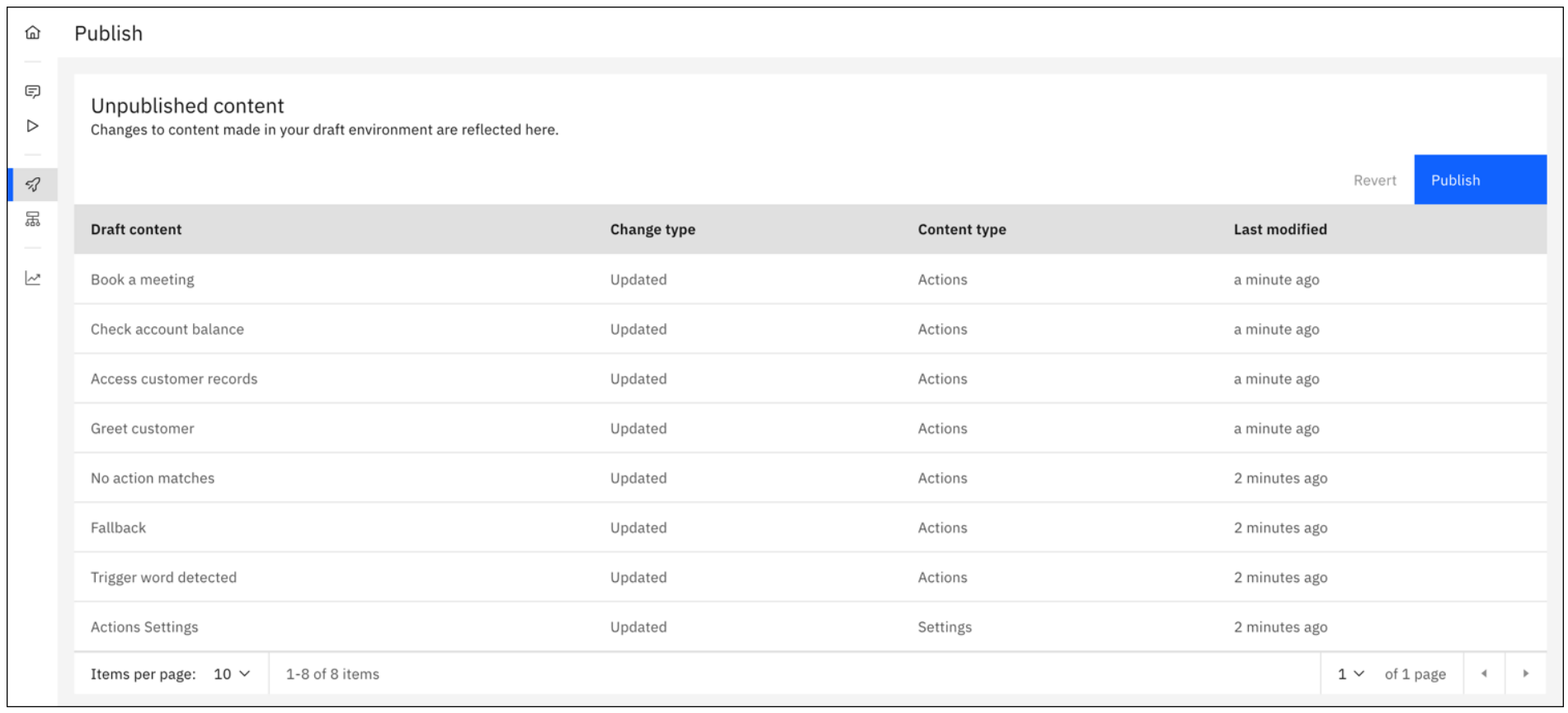
To remove the background image:

1. On the **Preview** page, click **Change background**.
2. Click **Remove background**, then click **Continue**.
3. Click **Remove background** to finish.

Publishing your content

Publishing is a way to maintain a healthy lifecycle management process. You can create incremental versions of your content over time, making it easier to manage deployment of changes and roll back (revert) to prior versions if necessary.

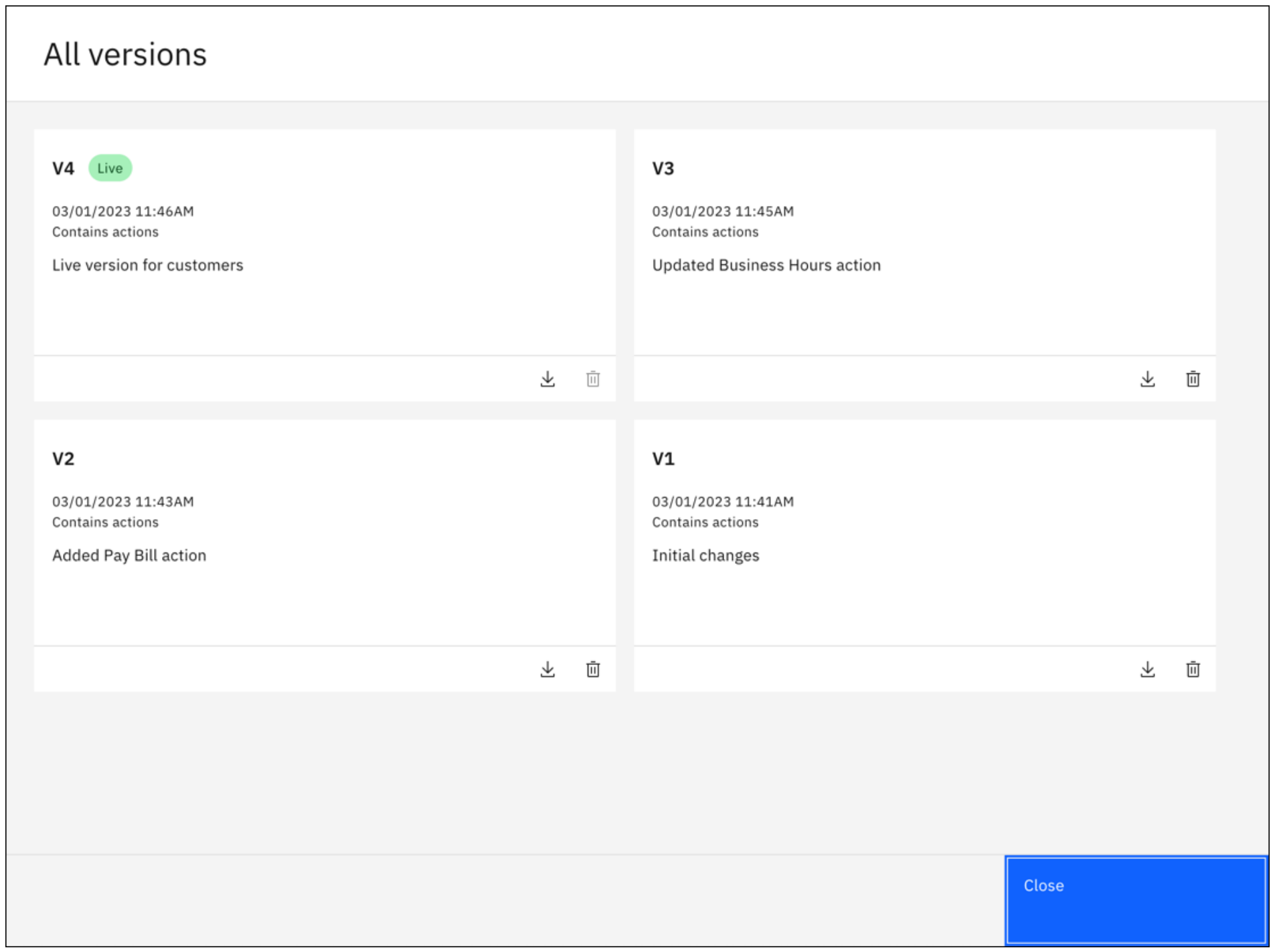
When you are ready to create a snapshot of your content and settings, you can publish from the **Publish** page. Each time that you publish, you create a new version, such as V1 or V2.



Publish page screen.

When you publish your content, watsonx Assistant creates a snapshot of the draft content, resulting in a version. This version contains all of the content from actions, including settings and variables. Versions do not contain integration configurations or environment settings. Integration configurations and environment settings must be configured manually in each environment.

The three most recent published versions appear in a list on the **Publish** page itself. If you have more than three versions, you can click **View all** to see a list of all published versions.



All versions

The number of versions that can be maintained depends on the type of plan you have. If you reach the plan limit of versions you can have, you need to delete a version before you can publish another one.

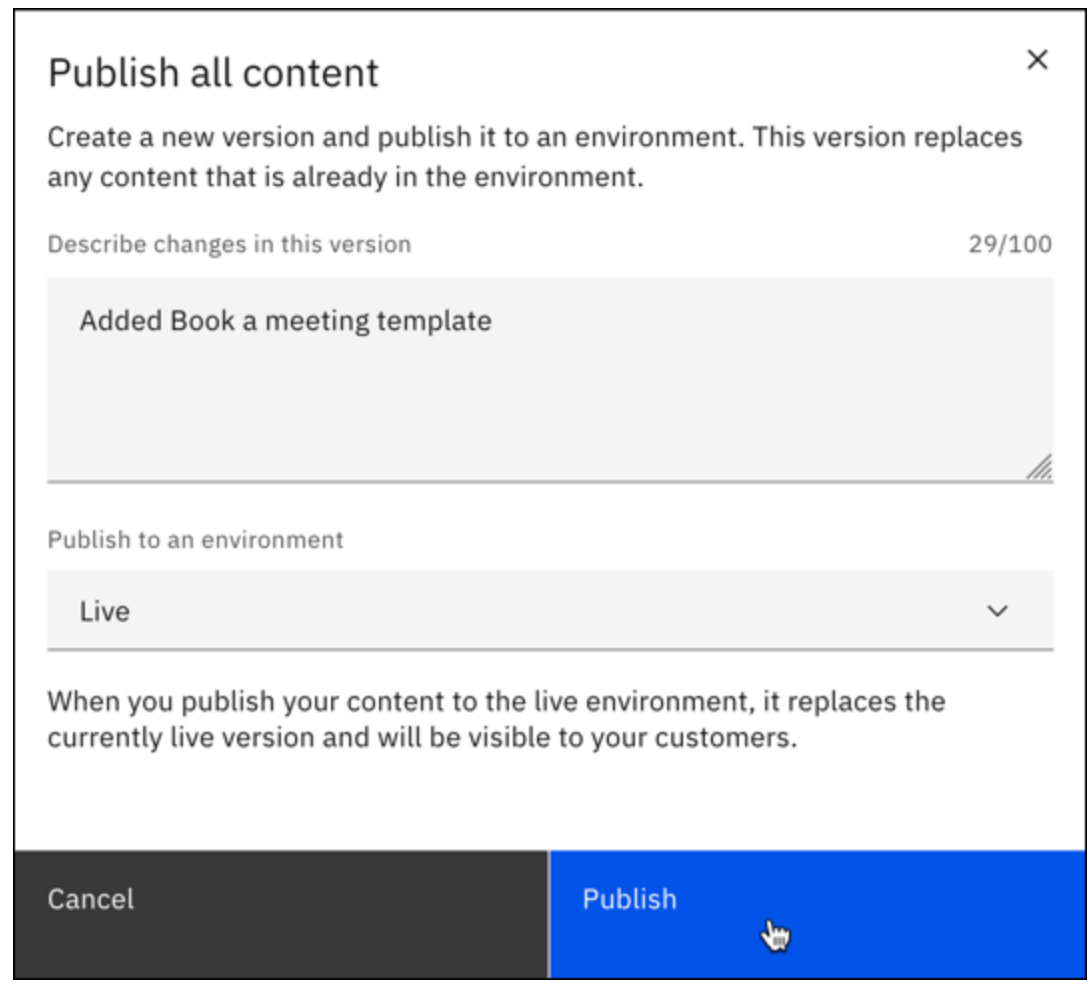
Service plan	Published versions
Enterprise	50
Premium (legacy)	50
Plus	10
Trial	10
Lite	2

Service plan published versions

How to publish

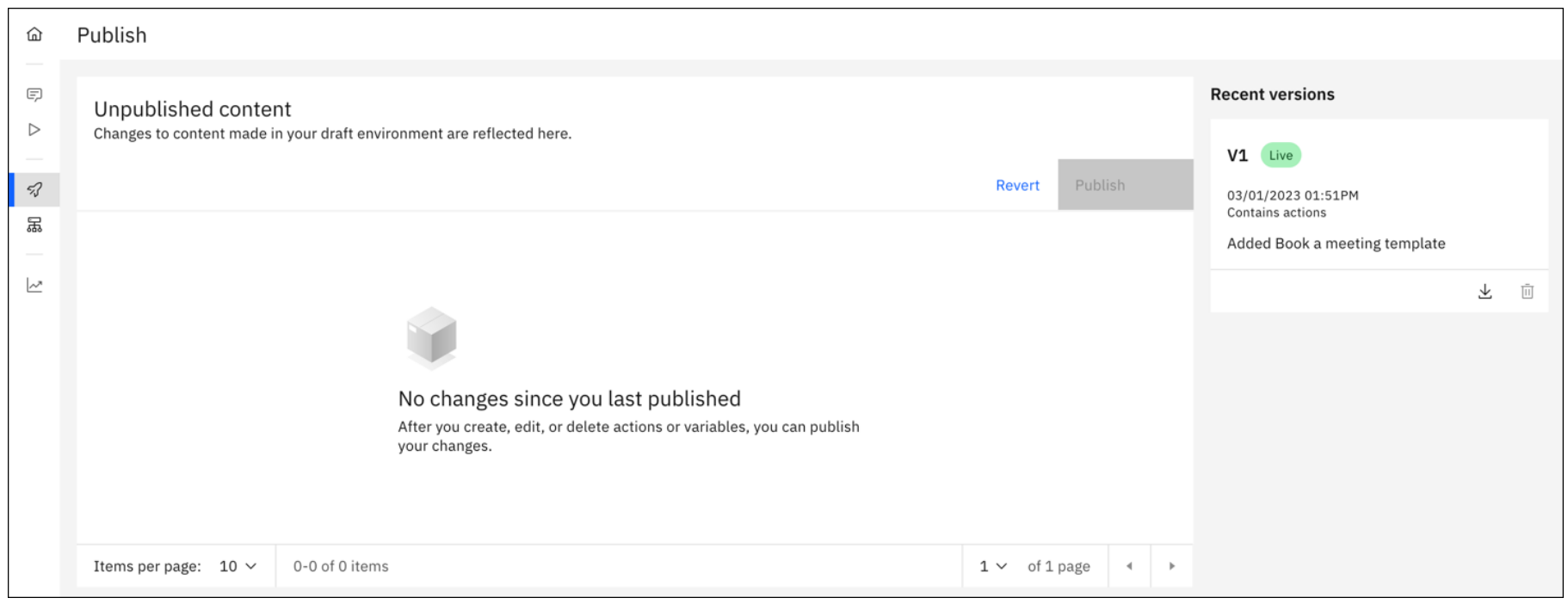
1. If changes are available to publish, click **Publish**.
2. Enter a description of the version.
3. Choose to publish to the live environment, or to create a version without publishing. If you choose to create a version, you can use the environment tab to publish that version later.

   With the Enterprise plan, you can publish to an environment you added. For more information, see Adding and using multiple environments.
4. Click **Publish**:



Publish all content

5. After you publish, the list is clear until you make more edits to your content.



No changes

What is being published?

When you publish, a snapshot is taken of the current state of actions. All updates to actions content are published in a version, including the following updates:

- Creating an action
- Editing an action
- Editing actions settings
- Deleting actions
- Creating variables
- Editing variables
- Deleting variables

If you are using dialog with actions, these updates are published:

- Activating dialog
- Creating a dialog
- Editing a dialog
- Editing dialog settings
- Deleting a dialog

The following updates are not published in a version and must be configured manually for each environment:

- Channel configurations
- Environment settings

Publishing or switching versions in the live environment

There are two ways to publish a version to the live environment:

- When you publish a version from the draft environment
- Use the **Live environment** tab and click **Publish version**.

After a version is published to the live environment, there are two ways you can switch to a different version:

- When you publish a version from the draft environment, you have the option of assigning it to the live environment, replacing the one that's already there
- Use the **Live environment** tab and click **Switch version**.

Reverting to a previous version

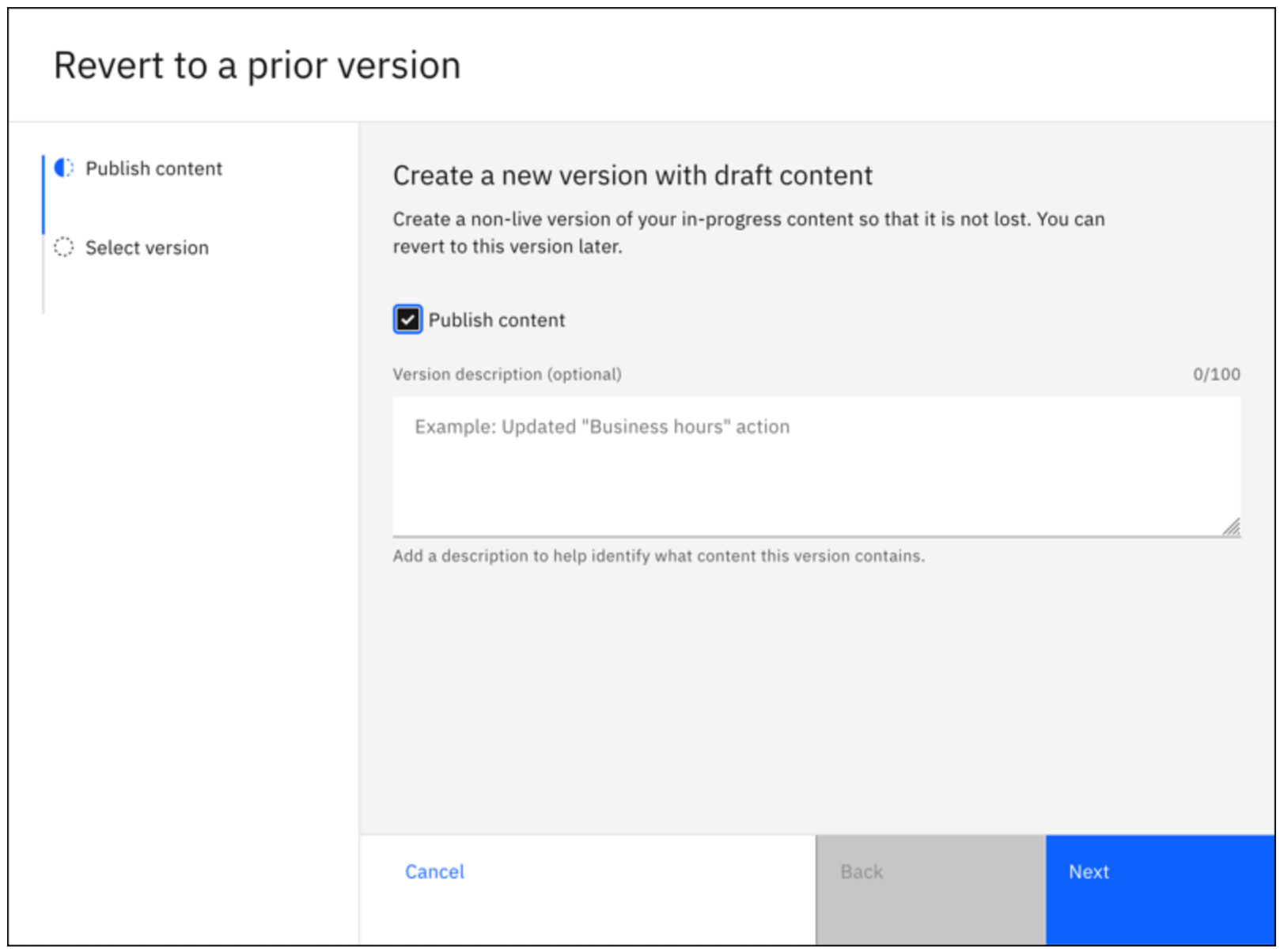
Use the revert function if you need to update a version of content that is already published to a version. For example, if you publish V1 of your content and then notice an error, you can revert your actions to V1 and use the draft environment to correct the error.

If you revert to a previous version, your published content isn't affected, but the content on the **Actions** page is overwritten with the version that you revert to. For example, if you decide to revert to V1 of your content, all content on the **Actions** page is overwritten with the V1 content so you can continue to work on it.

To revert to a previous version:

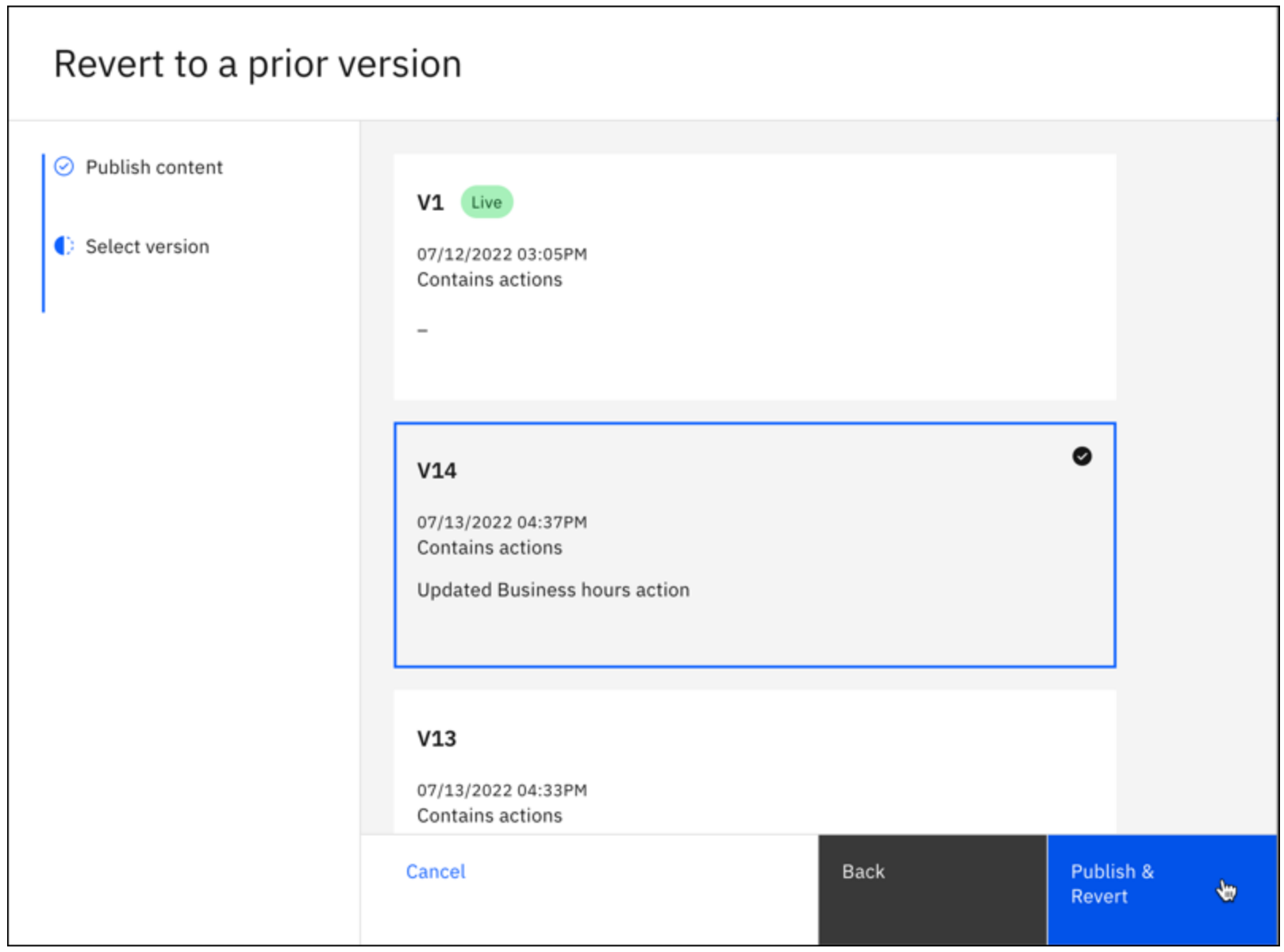
1. On the **Publish** page, click **Revert**.
2. On the **Publish content** tab, choose whether you want to save your in-progress actions as a content version so that you don't lose your current work. For example, if you are reverting to V1 and you choose to create a version, V2 is created so you can resume work on it later.

   To create a version, check **Publish content** and add a description.



Publish content

3. Next, select the version of content that you want to revert to, then click **Publish & Revert**.



Revert to a prior version

4. After you revert, you can go to the **Actions** page to make any fixes or updates. When you're ready to publish, go to the **Publish** page and publish your updated content as a new version (for example, V3).

Adding and using multiple environments

Enterprise

Each assistant has a draft and live environment. For Enterprise plans, you can add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments.

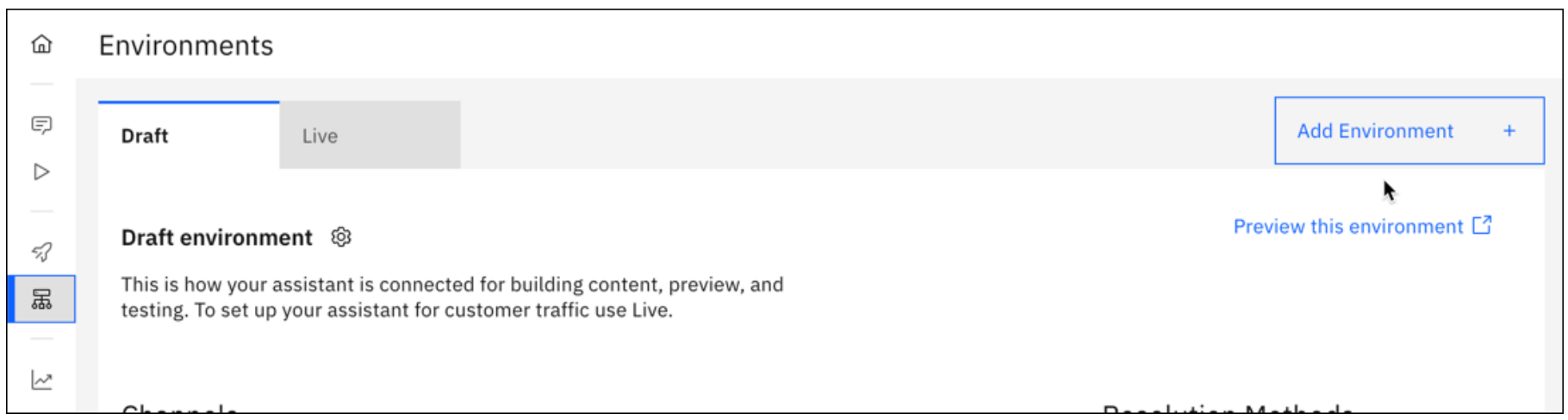
For more information about environments, see Environments.

Adding environments

Each new environment appears as an extra tab on your **Environments** page. You can't reorder these environments, so add them to match your test-to-deploy lifecycle.

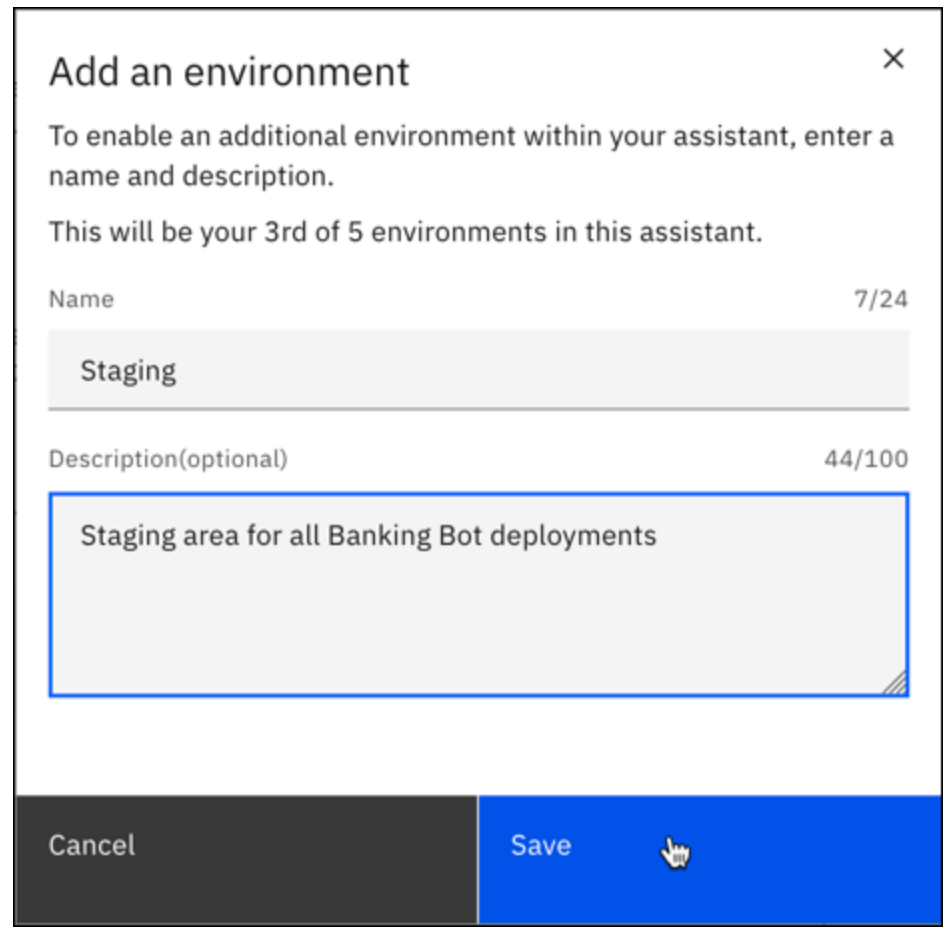
To add an environment:

1. Open the **Environments** page and click **Add Environment**.



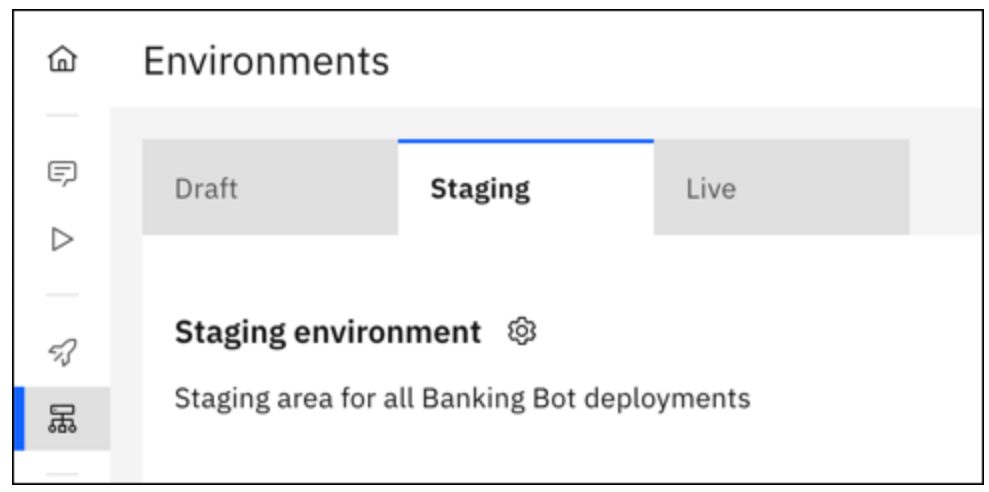
Add environment

2. Enter a name and a description, then click **Save**. Names can't contain spaces or use any special characters.



Add an environment

The new environment appears as an extra tab on your **Environments** page.



Environment tab

Using your environments

You can use extra environments in your development and test process before you deploy an assistant for customer use.

Add extra environments to match your existing test-to-deploy process. For example, you might name and use five environments in a scenario like this:

Environment	Used for
Draft	Conversation authors build actions and test as they work
Review	Conduct an initial content review with stakeholders
Test	Test assistant content on a configured channel
Staging	Use a staging website to test assistant content and channel configurations
Live	Deploy for customer use

Example

Access control to environments

You can control who can work in each environment. Each environment has an ID that you can use in IBM Cloud Identity and Access Management (IAM) and set access by resource. For more information on access control, see Managing access with Identity and Access Management. For more information on settings, see Environment settings.

Publishing content to an environment

For more information, see Publishing your content.

To publish new changes from the draft environment to an environment:

1. If changes are available to publish, click **Publish**.
2. Enter a description of the version.
3. Choose an environment.
4. Click **Publish**.

To publish an existing version to an environment:

1. On the **Environments** page, click the environment tab.
2. In Resolution Methods, click **Switch version**.
3. Choose a version to publish, then click **Switch version**.

Moving a version through multiple environments

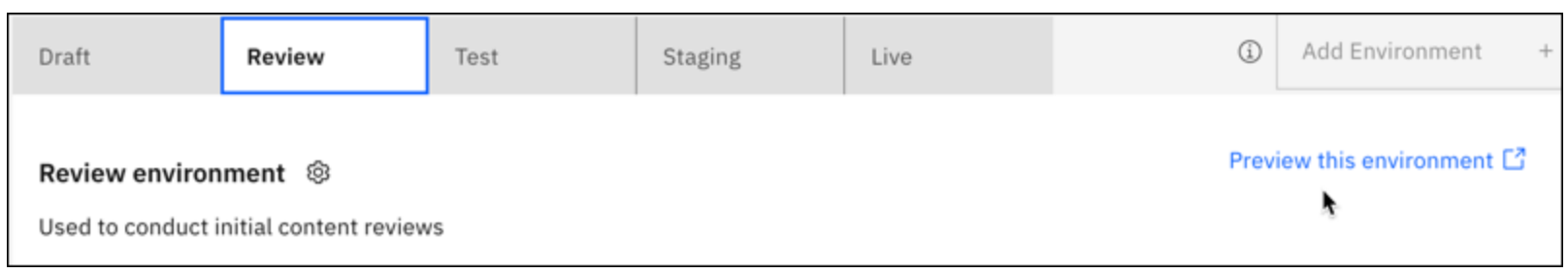
This example explains moving a content version through multiple environments to build, test, iterate, and deploy.

Environment	Activity
Draft	Publish version V3 to Review environment
Review	Conduct initial testing of V3
Test	Switch to version V3 for further testing with a configured channel
Staging	Switch to version V3 for testing with an internal staging website
Live	Switch to version V3 for customer use
Live	Switch to version V2 after a bug is found in version V3
Draft	Revert to version V3 to fix the bug. For more information, see Reverting to a previous version
Draft	Publish version V4 to Review environment for retesting
Test	Switch to version V4 for further retesting
Staging	Switch to version V4 for testing with an internal staging website
Live	Switch to version V4 for customer use

Example

Previewing an environment

On each environment tab, you can click **Preview this environment** to open another browser tab and preview your assistant as an interactive web chat widget.



Preview

You can share this unauthenticated version of your assistant with your team by sending them the link to the environment preview. Your subject-matter experts can test your in-progress assistant without needing access to watsonx Assistant itself.

Environment settings

Each environment has its own settings. Click the **Settings** gear icon to open the settings.

Setting	Description
API Details	Environment name and ID, session URL, and a link to the IBM Cloud console to see the service credentials for your instance
Webhooks	Settings for pre-message, post-message, and log webhooks. For more information, see Extending your assistant with webhooks.
Inactivity timeout	Specify the amount of time to wait after the customer stops interacting with the assistant before the session ends. The maximum inactivity timeout differs by service instance plan type. For more information, see Inactivity timeout.
Session history	For each environment, you can record the recent messages from the conversation for each customer, for use with the session_history variable. For more information, see Session history.
Edit environment	Change the name or description of the environment. Names can't contain spaces or use any special characters.
Delete environment	If necessary, you can remove the environment. Any channel or extension configurations are removed. Deleting an environment doesn't delete any published content versions. They remain in your list of published versions.

Example

Deploying to your website or mobile app

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

How the web chat works

The web chat provides an easy-to-use chatbot interface that you can add to your website without writing any code.

After you add the web chat script to your website, your customers see a launcher icon that they can click to open the chat window and start a conversation with the assistant. The appearance of the launcher icon adapts to desktop and mobile browsers.

When a customer clicks the launcher, the web chat window opens, initially displaying the *home screen*. The home screen displays a greeting and an optional set of suggested conversation starters for common questions and problems. The customer can either click a conversation starter or type a message in the input field to start the conversation with the assistant.

Tip: The appearance and behavior of the launcher icon, the home screen, and most other aspects of the web chat can be configured and customized to match your website style and branding. For more information, see Configuring the web chat.

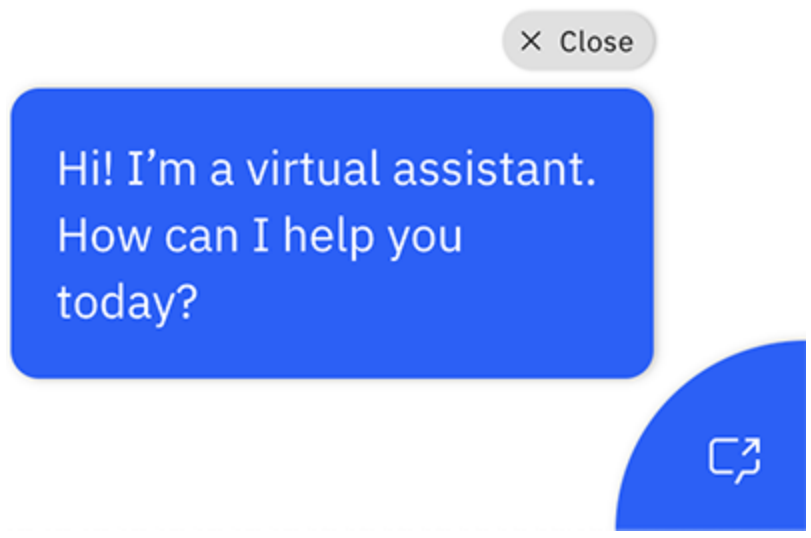
Launcher appearance and behavior

The web chat launcher welcomes and engages customers so they know where to find help if they need it. By default, the web chat launcher is displayed as a circle in the bottom-right corner:

After 15 seconds, the launcher expands to show a greeting message to the user. In this expanded state, a customer can still click the launcher to open the web chat. (If the customer reloads the page or navigates to a different page before the launcher expands, the 15-second timer restarts.)

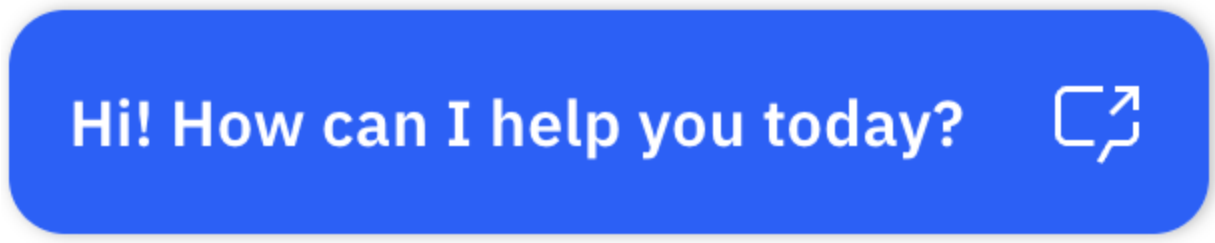
The appearance of this expanded state differs slightly depending on whether the customer is using a desktop browser or a mobile browser:

- For desktop browsers, the expanded launcher shows two primary buttons the customer can click to open the web chat, and a **Close** button that closes the launcher.



  The expanded launcher remains in its expanded state even if the customer reloads the page or navigates to a different page. It stays in its expanded state until the customer opens it. The customer can open it by clicking one of the two primary buttons. When the customer closes it, it returns to its initial small state for the rest of the session.

- For mobile browsers, the launcher shows only a single primary button.



  The customer can close the launcher by scrolling on the page, swiping right on the expanded launcher, or waiting 10 seconds, at which point the expanded launcher shrinks back to its initial small state automatically. If the user reloads the page or navigates to a different page while the launcher is expanded, it stays in its expanded state, and the 10-second timer restarts.

After the next page refresh, if the launcher remains in its small state without being clicked, it bounces up and down to attract the customer's attention. The first bounce happens 15 seconds after the page refresh; if the customer still does not click the launcher, it bounces again 60 seconds later. (The timing of the second bounce might be affected if the user refreshes the page or navigates to a different page.) If the user still does not click the launcher, it does not bounce again.

The language of the default text that is shown within the launcher depends on the locale that is configured for the web chat. If you customize the greeting text, the text you provide is used regardless of the locale settings.

You can configure the color of the launcher, and the greeting message text, in the web chat settings. For more information, see Configuring the web chat.

Rendering assistant output

In addition to plain text, watsonx Assistant supports many response types that can be used to output multimedia and interactive elements. The web chat includes built-in support for a wide variety of response types:

- **Text formatting**: The web chat supports formatting text that uses either Markdown or HTML. For more information, see Markdown formatting.
- **URLs**: Valid URLs (such as `http://example.com`) are automatically rendered as clickable links. When a customer clicks a link in the web chat, the target website opens in a new browser tab.
- **Options**: Options responses (when the assistant asks the customer to select from a set of choices) are automatically rendered as interactive elements. (By default, a set of fewer than five options is rendered as a set of clickable buttons; five or more options are rendered as a drop-down list.)
- **Dates**: When the assistant asks the customer to specify a date, the web chat displays an interactive date picker. The customer can specify the date either by clicking the date picker or by typing a valid date value in the input field.
- **Multimedia responses**: The web chat supports all multimedia response types (`audio` , `image` , and `video`).
- **iframe**: The web chat supports the `iframe` response type, which embeds HTML content (such as a form or interactive map) directly in the web chat window.

For more information about how the web chat handles specific response types, see the Response types reference.

Markdown formatting

In text responses from your assistant, you can use Markdown formatting to apply highlighting such as italics, or to include elements like paragraphs and headings. Some common examples of Markdown formatting include:

- Headings:

```
$ # First-level heading

## Second-level heading
```

- Highlighting:

```
$ This text includes *italic* and **bold** highlighting, as well as a `code` snippet.
```

- Lists:

```
$ 1. ordered
2. list

- bulleted
- list
```

- Tables:

```
$ | Column 1 | Column 2 |
|----------|----------|
| Row      | One      |
| Row      | Two      |
```

- Links:

```
$ [This link](https://www.ibm.com/products/watson-assistant/demos/lendyr/demo.html) opens in a new tab.

[This link](https://www.ibm.com/products/watson-assistant/demos/lendyr/demo.html){{target=\"_self\" rel=\"noopener noreferrer\"}} opens in the same tab.
```

For more information about the Markdown format, see the CommonMark specification.

Live agent transfer

The web chat supports transferring the customer to a human in situations the assistant can't handle. If you configure one of the supported contact center integrations, the web chat can open a separate chat window in which the customer can communicate with a live agent.

Your assistant can then initiate a transfer in situations when the assistant is unable to handle a customer's requests. (For more information about initiating a transfer, see Connecting to a live agent.)

For information about how to add a contact center integration to the web chat, see Adding contact center support.

Technical details

Web chat is displayed on your website by a short JavaScript code snippet, which calls additional JavaScript code that is hosted by IBM Cloud. The hosted code is automatically updated with new features and fixes, so by default you always have the latest version. (You can optionally lock to a specific version if you prefer to control upgrades yourself.)

The code snippet that creates the web chat widget includes a configuration object, which you can modify to change the appearance and behavior of the web chat. The configuration object also specifies details that enable the web chat to connect to your assistant. If you are comfortable writing JavaScript code, you can customize the web chat by modifying the code snippet and by using the web chat API.

The web chat uses the watsonx Assistant v2 stateful API to communicate with the assistant. By default, the session ends and the conversation ends after 5 minutes of inactivity. This means that if a user stops interacting with the assistant, after 5 minutes, any context variable values that were set during the previous conversation are set to null or back to their initial values. You can change the inactivity timeout setting in the assistant settings (if allowed by your plan).

Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

The web chat integration complies with the Web Content Accessibility 2.1 Level AA standard. It is tested with both screen readers and automated tools on a continual basis.

Billing

watsonx Assistant charges based on the number of unique monthly active users (MAU).

By default, the web chat creates a unique, anonymous ID the first time a new user starts a session. This identifier is stored in a first-party cookie, which remains active for 45 days. If the same user returns to your site and chats with your assistant again while this cookie is still active, the web chat integration recognizes the user and uses the same user ID. You are charged only once per month for the same anonymous user.

> ⚠ **Important:** On Apple devices, the Intelligent Tracking Prevention feature automatically deletes any client-side cookie after 7 days. If an anonymous customer accesses your website and then visits again two weeks later, the two visits are treated as two different MAUs. For information about how to prevent this problem, see Managing user identity information in web chat.

For information about how to customize the handling of user identity information for billing purposes, see Managing user identity information in web chat.

The usage is measured differently depending on the plan type. For Lite plans, usage is measured by the number of `/message` calls (API) are sent to the assistant from the web chat integration. For all other plans, usage is measured by the number of monthly active users (MAU) that the web chat interacts with. The maximum number of allowed MAUs differs depending on your watsonx Assistant plan type.

Plan	Maximum usage
Enterprise	Unlimited MAU
Premium (legacy)	Unlimited MAU
Plus	Unlimited MAU
Trial	5,000 MAU
Lite	10,000 API (approximately 1,000 MAU)

Plan details

Browser support

The web chat supports various devices and platforms. Generally, if the last two versions of a browser account for more than 1% of all desktop or mobile traffic, the web chat supports that browser.

The following list specifies the minimum required browser software for the web chat (including the two most recent versions, except as noted):

- Apple Safari
- Apple Safari Mobile
- Google Chrome
- Google Chrome for Android
- Microsoft Edge (Chromium and non-Chromium)
- Mozilla Firefox
- Mozilla Firefox ESR (most recent ESR only)
- Mozilla Firefox Mobile
- Opera
- Samsung Mobile Browser
- UC Browser for Android

For optimal results rendering the web chat on mobile devices, the `<head>` element of your web page must include the following metadata element:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Cookies and GDPR

Web chat stores an ID to identify the current session for the length that the browser is open. This allows web chat to keep the session open as users navigate your page. Once a user closes the browser, the information is removed.

- If you supply web chat with a userID, the only data or cookies stored on the browser are from the current session. This userID can be used to delete the user's data on request, as it is also passed as the `customer_id` portion of the X-Watson-Metadata HTTP header, which is why the userID syntax must meet requirements for header fields defined in RFC 7230 (all visible ASCII characters). More about removing user data at Securing your assistant.

- If you fail to supply web chat with a unique `userID`, we add a first-party cookie with a generated anonymous ID. The cookie remains active for 45 days to ensure we count a user once, even with multiple visits in the same month, and not pollute billing metrics.

Web chat keeps analytics on what features are used so we can continually improve user experience. None of the data collected identifies an end user to IBM, and users are not tracked across different web sites. We only make note of behavior within web chat.

Language support

By default, the web chat displays hardcoded labels and messages in English, but support is built in for all of the languages that are supported by watsonx Assistant. You can also choose from a wide selection of locales to customize the display of strings like dates and times for global audiences.

In whichever language you are using, you can also customize the text of any hardcoded strings.

For more information, see Supporting global audiences in web chat.

Security

By default, all messages that are sent between the web chat and the assistant are encrypted with Transport Layer Security (TLS). You can enable the web chat security feature if you need more robust protection.

The web chat embed script that you include on your website contains unique identifiers (such as the integration ID and service instance ID) that enable the web chat to connect with your assistant. These identifiers are not considered secret, and are visible to anyone who has access to your website. Anyone who has these IDs can use them to send messages to your assistant and receive its replies. However, these IDs cannot be used to log in to your account, make changes to your assistant, or retrieve logs or analytics information about your assistant.

If you are concerned about unauthorized access to your assistant, you can enable the web chat security feature for extra security, such as verifying message origin and authenticating users. Enabling the security feature requires more development work on your website. For more information, see Web chat security.

Updating site security policies

If your website uses a Content Security Policy (CSP), you must update it to grant permission to the web chat.

Table. CSP properties lists the values to add to your CSP.

Property	Values
default-src	'self' *.watson.appdomain.cloud 'unsafe-inline'
connect-src	*.watson.appdomain.cloud

CSP properties

Note: For default-src, if you have a separate `font-src`, you must add `*.watson.appdomain.cloud` along with `font-src`.

The following example shows a complete CSP metadata tag:

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self' *.watson.appdomain.cloud fonts.gstatic.com 'unsafe-inline';connect-src *.watson.appdomain.cloud" />
```

Allowing elements

If your CSP uses a nonce to add elements such as `<script>` and `<style>` tags to an allowlist, do not use `unsafe-inline` to allow all such elements. Instead, provide a nonce value to the web chat widget as a configuration option. The web chat sets the nonce on any of the `<script>` and `<style>` elements that it generates dynamically.

A CSP that passes a nonce to the web chat widget might look like this:

```
$ <meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self' *.watson.appdomain.cloud fonts.gstatic.com 'nonce-<server generated value>';connect-src *.watson.appdomain.cloud"
>
```

You can pass the nonce to the web chat by editing the embed script as follows:

```
window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE",

  cspNonce: "<server generated value>",

  onLoad: function(instance) {
    instance.render();
  }
};
```

Access to web chat hosts

If the system that hosts your website has limited Internet access (for example, if you use a proxy or firewall), make sure the following URLs are accessible:

- `https://web-chat.global.assistant.watson.appdomain.cloud`: Hosts the code for the web chat widget, and is referenced by the script you embed on your website.
- `https://integrations.{location}.assistant.watson.appdomain.cloud`: Hosts the web chat server, which handles communication with your assistant. Replace `{location}` with the location of the data center where your service instance is located, which is part of the service endpoint URL. For more information, see Finding and updating the endpoint URL.

Reviewing security

The web chat integration undergoes tests and scans regularly to find and address potential security issues, such as cross-site scripting (XSS) vulnerabilities.

Be sure to run your own security reviews to see how the web chat fits in with your current website structure and policies. The web chat is hosted on your site and can inherit any vulnerabilities that your site has. Serve content over HTTPS, use a Content Security Policy (CSP), and implement other basic web security precautions.

Copying session state

The web chat integration stores the state of the current session as cache in the user's web browser. When the user sends a query to the assistant, the assistant replies along with a copy of the current state of the session. When the assistant session expires in the server due to inactivity timeout, web chat creates a new session and copies the previous session state to the new session. Therefore, the users get a seamless conversation from the user if the session expires while the user is in the middle of a conversation. This state is only preserved as long as the user keeps the browser tab with web chat open. If the user reloads the page, leaves the page, or closes the tab, this session state is lost and they will get a new session with new state the next time they open web chat. When the session on the server expires, the user is presented with a warning that they will need to send a message to continue the conversation.

Note: The session state stored by web chat does not include any private variables. These variables are stored only in the server. When an assistant copies the old session state from the server in a new session, which started after the expiry of the old server session, any private variables that were set from the old session are lost.

Web chat setup

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.

Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Configuring style and appearance

Steps to Customize the Chat Window

You can configure the style and appearance of your web chat widget by doing the following steps.

1. Go to Home > Preview > Customize web chat > Style to configure the overall appearance of the web chat widget.
2. Select the name of the assistant per the customer requirement so that it is displayed in the header of the chat window. The name can be up to 64 characters in length.
3. You can set the following colors:

- Primary color: The color of the web chat header.
- Secondary color: The color of the customer input message bubble.
- Accent color: The color of the interactive elements, including:
    - Web chat buttons, such as the suggestions button and the **send message** button.
    - The border around the input text field (when in focus).
    - The markers that appear next to the assistant's messages.
    - The borders around the buttons and drop-down lists when customers select from options.
    - The **home screen** background.

> **Tip:** Each color is specified as an HTML hexadecimal color code, such as `#FF33FC` for pink and `#329A1D` for green. To change a color, type the HTML color code that you want to use, or click the dot next to the field and choose the color from the interactive **color picker**.

1. Toggle the **IBM Watermark** switch to `enable` or `disable` the `Built with IBM Watson` watermark in the web chat widget.

> **Note:** The watermark cannot be disabled if you are on the Lite plan of watsonx Assistant.

2. Click **Add an avatar image** to add an image, or **Change avatar image** to change an existing image. Specify the URL for a publicly accessible hosted image, such as a company logo or assistant avatar. The image must be between `64×64` and `100×100` pixels in size.

Style changes are immediately reflected in the web chat preview shown on the page. However, no configuration changes are applied to the environment until you **Save and exit**.

You can now use the Carbon for AI in your web chat style. For more information about Carbon for AI integration in your web chat, see Configuration.

You can customize a full-screen chat window by setting a max-width and removing the drop shadow that provides seamless integration for your assistant. For more information, see Layout configuration.

Configuring the home screen

On the **Home screen** tab, you can configure the contents of the home screen, which welcomes customers and helps them start the conversation with the assistant. The home screen replaces any greeting that would otherwise be sent by the *Greet customer* system action.

If you prefer to use a *Greet customer* system action instead of the home screen, you can disable it by clicking the toggle on the **Home screen** tab.

If you use the home screen, you must configure it to show content that is relevant to your customers:

- In the **Greeting message** field, type a greeting that is engaging and invites the user to interact with your assistant. This greeting is the first message that your customers see when they open the web chat window.
- In the **Conversation starters** section, specify the conversation starter messages that you want to be displayed on the home screen.

  These messages are displayed on the home screen as options that customers can click to start the conversation (for example, `I need to reset my password` or `What is my account balance?`). Specify conversation starters that are likely to be useful to your customers, and that your assistant knows how to handle.

  Be sure to test each conversation starter. Use only messages that your assistant understands and knows how to answer well. Conversation starters cannot be longer than 35 characters.

  You can specify up to 5 conversation starters.

The messages that you specify are immediately reflected by the web chat preview that is shown on the page, and you can click the conversation starters to see how your assistant responds. However, no configuration changes are applied to the environment until you click **Save and exit**.

Configuring suggestions

Suggestions give your customers a way to try something else when the current exchange with the assistant isn't delivering what they expect. A question mark icon ? is displayed in the web chat that customers can click at any time to see other topics that might be of interest or, if configured, to request support. Customers can click a suggested topic to submit it as input or click the **X** icon to close the suggestions list.

If customers select a suggestion and the response is not helpful, they can open the suggestions list again to try a different suggestion. The input generated by the first choice is submitted and recorded as part of the conversation. However, any contextual information that is generated by the initial suggestion is reset when the subsequent suggestion is submitted.

The suggestions are shown automatically in situations where the customer might otherwise become frustrated. For example, if a customer uses different wording to ask the same question multiple times in succession, and the same action is triggered each time, then related topic suggestions are shown in addition to the triggered action's response. The suggestions that are offered give the customer a quick way to get the conversation back on track.

The suggestions list is populated with actions that are relevant in some way to the matched action. The actions are ones that the AI model considered to be possible alternatives, but that didn't meet the high confidence threshold that is required for an action to be listed as a disambiguation option. Any action can be shown as a suggestion, unless its **Ask clarifying question** setting is set to **Off**. For more information about the **Ask clarifying question** setting, see Asking clarifying questions.

To configure suggestions, complete the following steps:

1. Open the **Suggestions** tab.

   Suggestions are enabled automatically for new web chat integrations. If you don't want to use suggestions, toggle the switch to **Off**.

2. In the **Include a connection to support** section, specify when you want an option to connect with support to be included in the list of suggestions. You can specify **Always**, **Never**, or **After one failed attempt**.

   After one failed attempt: Adds the option to the list only if the customer reached a node with an `anything_else` condition in the previous conversation turn or reaches the same action for a second time in succession.

3. In the **Option label** field, type the text of the message that requests help from support. This message is shown as the label for the support option, which is included in the **Suggestions** window under the circumstances you specified in the previous step. If the customer clicks this option, the same message is sent to the assistant.

   The message you specify should trigger an action that gives customers a way to connect with support. By default, the message `Connect with agent` is used. If your web chat is integrated with a contact center platform, this message initiates a transfer to a human agent. (For more information about integrating with a contact center, see Adding contact center support.)

   If your web chat is not integrated with a contact center, specify a message that helps your customers reach whatever form of support you do offer. If you offer a toll-free support line, you might add `Get the support line phone number`. Or if you offer an online support request form, you might add `Open a support ticket`.

   Whether you use the default support message or add your own, make sure your action is designed to recognize the message and respond to it

appropriately.

Web chat security

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Enabling web chat security

To enable web chat security, you must make changes to your web application server code and the web chat embed script, as well as the web chat integration settings.

Note: For a complete, NodeJS reference implementation enabling security, see Enabling security for watsonx Assistant web chat.

Before you begin

Before you enable security, you must create an RS256 public/private key pair. You can use a tool such as OpenSSL or PuTTYgen.

For example, to create the key pair at a command prompt using OpenSSL, you would use the command `openssl genrsa -out key.pem 2048`.

Save the generated key pair in a secure location.

Important: Make sure these keys are accessible only by your server code. Never pass them to a client browser through your website.

Generating a JWT

To use web chat security, you must configure the web chat on your website to send a JSON Web Token (JWT) with each message to the assistant. The JWT

is used to verify the origin of messages sent from your website, and optionally to carry additional encrypted data. Your website will need to be able to generate a new JWT at the beginning of each session, and also whenever an existing JWT expires.

> ⚠ **Important:** Do not hardcode a JWT in your website code or share JWTs between users.

On your server, implement a function that generates and returns a JSON Web Token (JWT) that is signed with your private key. You will use this token to verify the origin of messages sent from your website, and optionally to carry additional encrypted data.

Most programming languages offer JWT libraries that you can use to generate a token. To validate signed JWTs, the web chat integration uses the jsonwebtoken library with the `RS256` algorithm.

The JWT payload must specify the following claims:

- `sub`: A unique user ID that identifies the customer who is interacting with the web chat. This can be either a generated unique identifier (for anonymous users) or an authenticated user ID. For more information about how the `sub` value is used, see Authenticating users in web chat.

  To ensure security, the JWT should be specific to each user. Use either the user's authenticated login information, or a unique generated ID. Do not reuse the same JWT, or the same `sub` value, for more than one user.

- `exp`: The expiration time after which the JWT is no longer valid. Many JWT libraries set this value for you automatically. Set a short-lived `exp` claim (for example, `1h`).

> **Tip:** Do not create a JWT without an `exp` claim. Although this claim is not formally required, omitting it represents a security exposure because anyone with access to the JWT could copy it and use it later to access your assistant. Setting an expiration time limits this exposure.

The following JavaScript example shows how you might generate a JWT using the `jsonwebtoken` library:

```
// Sample NodeJS code on your server.
const jwt = require('jsonwebtoken');

// Returns a signed JWT signed with the RS256 algorithm.
function createJWT() {
  const payload = {
    sub: 'some-user-id', // Identifies user for billing purposes
  };
  // The "expiresIn" option adds an "exp" claim to the payload.
  return jwt.sign(payload,
    process.env.YOUR_PRIVATE_RSA_KEY,
    { algorithm: 'RS256', expiresIn: '1h' });
}
```

Configuring the web chat to include JWTs

Now that you have implemented a function to generate a signed JWT, you must update your web chat instance to include the signed JWT with each message it sends. After you enable web chat security, any messages that are not signed with the proper private key are rejected.

In your website HTML, update the web chat embed script to specify a new JWT at the beginning of each session, and also whenever the existing JWT expires. The simplest way to do this is to subscribe to the identityTokenExpired event and generate a new JWT when that event is received. The `identityTokenExpired` event is fired in both of the following situations:

- At the beginning of a new session, if no JWT was provided using the `identityToken` configuration option.
- When the previously specified JWT expires.

In your `onLoad` event handler, use the on() instance method to subscribe to the `identityTokenExpired` event.

In the callback, call the function on your server that you implemented to generate a new JWT. Then use the `identityToken` parameter of the event to specify the new JWT, as in this example:

```
instance.on({ type: 'identityTokenExpired',
              handler: async function(event) {
                const jwtFromServer = await fetch('http://example.com:3001/createJWT');
                event.identityToken = jwtFromServer;
}});
```

The new JWT you specify is automatically sent with each subsequent message from the web chat instance on your web site (until the token expires).

Note: You can also specify the JWT at the beginning of the session by using the `identityToken` property in the web chat configuration object. However, you will still need to create an event handler for `identityTokenExpired`, unless you are certain your JWTs will never expire during a session.

Updating the web chat settings

Now that you have configured the web chat to send signed JWTs, you can enable web chat security in the web chat integration settings.

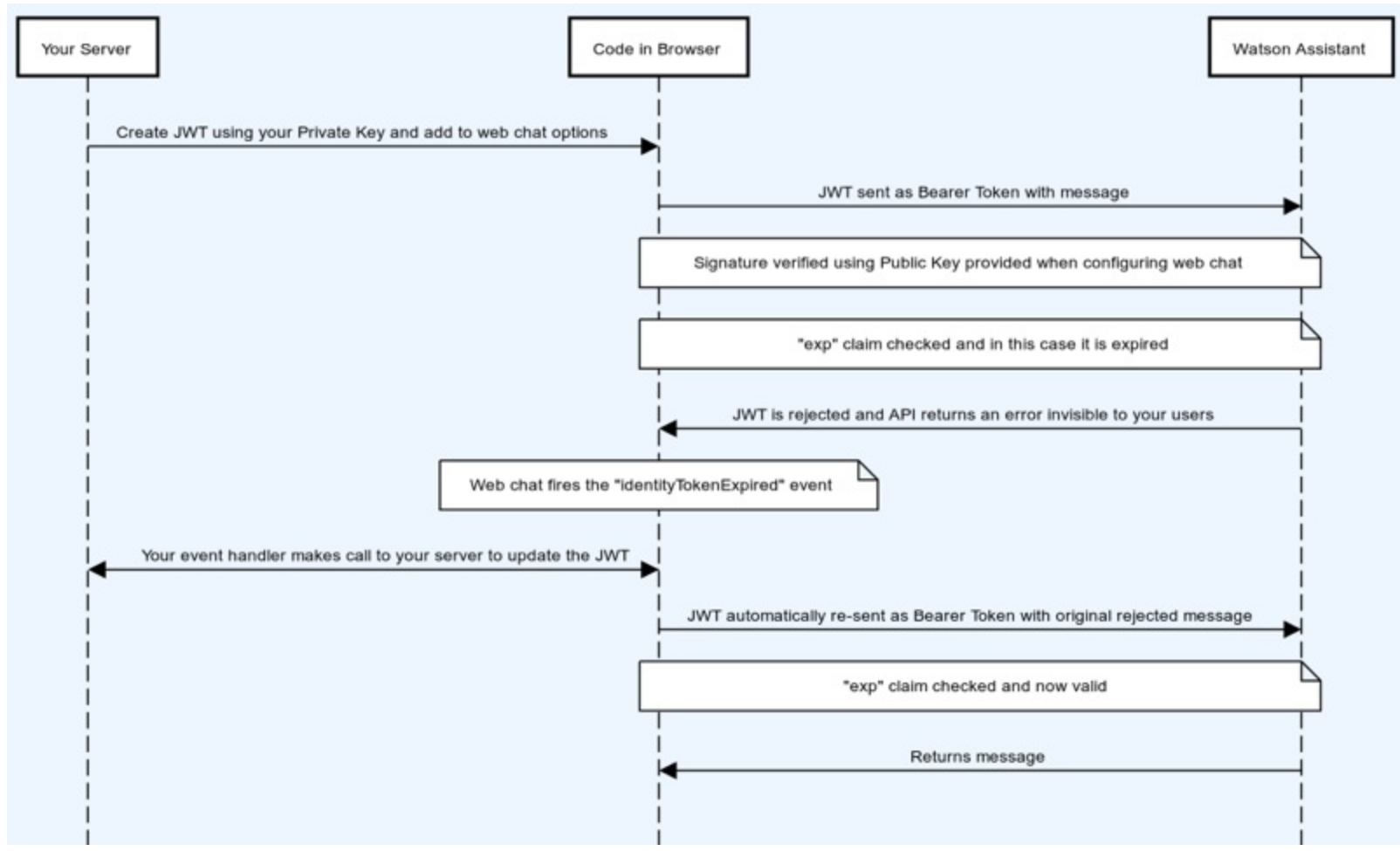
Important: After you enable web chat security, messages from any origin other than your web chat instance (which you have configured to send signed JWTs) are rejected. This means that enabling web chat security disables the shareable preview link, which does not send JWTs with messages. For more information about the preview link, see Copying a link to share.

To enable security, complete the following steps:

1. On the **Security** tab of the web chat integration settings, set the **Secure your web chat** switch to **On**.
2. In the **Your public key** field, paste your public key.

   watsonx Assistant uses the public key to verify that incoming messages originate from your website.

The following diagram shows the flow of messages between the web chat and the assistant when web chat security is enabled:



Authenticating users in web chat

With web chat security enabled, you can securely authenticate customers by user ID.

The default behavior of the web chat integration is to identify unique users by setting the value of the `user_id` property that is sent as part of each message to the assistant. For more information, see Managing user identity information in web chat.

This approach is sufficient for tracking unique users for billing purposes, but it is not secure and should not be used for access control. If you enable web chat security, use JSON Web Tokens (JWTs) to securely authenticate your users and control access to functions of your assistant that require authorization.

Authenticating with the `sub` claim

To use this method for authenticating users, you must first enable the web chat security feature. For more information, see Enabling web chat security.

When you create a JWT for the web chat, you must specify a value for the `sub` (subject) claim, which identifies the user. For anonymous users, you can use a generated unique ID.

Tip: When you generate a user ID for an anonymous user, be sure to save the generated ID in a cookie to prevent being billed multiple times for the same customer.

When the web chat integration receives a message signed with this JWT, it stores the user ID from the `sub` claim as `system_integrations.channel.private.user.id` in actions and `integrations.channel.private.user.id` for dialog. For more information, see Integration variables.

If your customers are required to log in before starting a web chat session, you can use the authenticated user ID as the value of the `sub` claim when you create the JWT. Because the web chat integration validates the JWT and uses the `sub` claim to set the user ID, your assistant can now rely on `system_integrations.channel.private.user.id` in actions and `integrations.channel.private.user.id` in dialog for secure access control to functions that require authorization.

Note: After you specify the JWT for the web chat, you cannot change to a JWT with a different `sub` claim during the session. If you need to add authenticated login information in the middle of a session, you can store it as part of the user payload instead. For an example of how to do this, see Tutorial: Authenticating a user in the middle of a session .

Billing and privacy

For user-based plans, the user ID is used for billing purposes. (You cannot use the `updateUserID()` instance method to set the user ID if web chat security is enabled.) The same user ID is also used as the customer ID, which can be used to make requests to delete user data. Because the customer ID is sent in a header field, the ID you specify must meet the requirements for header fields as defined in RFC 7230.

Important: If you are required to comply with GDPR requirements, you might need to persistently store any generated anonymous user IDs, especially for anonymous users who later log in with user credentials. Storing these user IDs makes it possible for you to later delete all data associated with an individual customer if requested to do so.

For more information about user-based billing, see User-based plans explained. For more information about deleting user data, see Labeling and deleting data.

Logging out

To log out a customer, you must destroy the web chat.

If you reload the page when a customer logs out, call the `destroySession()` instance method to remove any reference to the current session from the browser's cookies and storage. If you do not call this method, information that is protected by the JWT is not at risk, but the web chat will try to connect to the previous session and fail.

If you do not perform a full page reload when a customer logs out, call the `destroy()` instance method. The `destroy` method removes the current instance of the web chat that is configured for the current userID from the DOM and browser memory. Next, call the `destroySession()` instance method.

Encrypting sensitive data in web chat

By using the public key that is provided by IBM, you can add another level of encryption to hide sensitive data you send from the web chat.

Note: To use this method for encrypting data, you must first enable the web chat security feature. For more information, see Enabling web chat security.

Use this method to send sensitive information in messages that come from your website, such as information about a customer's loyalty level, a user ID, or security tokens to use in webhooks that you call from actions. Information passed to your assistant in this way is stored in a private context variable. Private variables cannot be seen by customers and are never sent back to the web chat.

For example, you might start a business process for a VIP customer that is different from the process you start for a standard customer. You do not want non-VIPs to know that they are categorized as such, but you must pass this information to your action so it can change the flow of the conversation. To do this, you can pass the customer MVP status as an encrypted variable. This private context variable is available for use by the action, but not by anything else.

Tip: **Tutorial:** For a tutorial that shows an example of using web chat security to authenticate users and protect sensitive data, see Tutorial: Authenticating a user in the middle of a session.

To encrypt sensitive data:

1. On the **Security** tab of the web chat integration settings, click the **Generate key** button.
2. Copy the public key that displays in the **IBM-provided public key** field. This field is available only if web chat security is enabled.
3. In the JavaScript function you use to create your JWT, include in the payload a private claim called `user_payload`. Use this claim to contain the sensitive data, encrypted with the IBM public key.

   The following code snippet shows a function that accepts a userID and user payload. If a user payload is provided, its content is encrypted and signed with the IBM public key. In this example, the public key is stored in an environment variable.

```
// Example code snippet to encrypt sensitive data in JWT payload.
const jwt = require('jsonwebtoken');
const RSA = require('node-rsa');

const rsaKey = new RSA(process.env.PUBLIC_IBM_RSA_KEY);

/**
* Returns a signed JWT. Optionally, also adds an encrypted user payload
* as stringified JSON in a private claim.
*/
function mockLogin(userID, userPayload) {
  const payload = {
    sub: userID, // Required
    // The exp claim is automatically added by the jsonwebtoken library.
};
if (userPayload) {
    // If there is a user payload, encrypt it using the IBM public key
    // and base64 format.
    payload.user_payload = rsaKey.encrypt(userPayload, 'base64');
}
const token = jwt.sign(payload, process.env.YOUR_PRIVATE_RSA_KEY, { algorithm: 'RS256', expiresIn: '1h' });
return token;
}
```

Tip: The user payload must be valid JSON data. The web chat uses the default options of the Node-RSA library, which expects the encryption scheme `pkcs1_oaep` and the encryption encoding `base64`.

4. When the web chat integration receives a message signed with this JWT, the content of the `user_payload` claim is decrypted and saved as the `context.integrations.chat.private.user_payload` object. Because this is a private variable, it will not be included in logs.

Web chat development

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.

Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Embedding the web chat on your page

To add the web chat widget to your website, you need to embed a generated script element in your HTML source.

The web chat integration is automatically included for every assistant, and is configured separately for each environment.

Note: For the webchat integration with a React-based application, see the alternative ways here.

To add the web chat to your website:

1. On the **Integrations** page, find the **Web chat** tile and click **Open**. The **Open web chat** window opens.
2. In the **Environment** field, select the environment you want the web chat widget to connect to. Click **Confirm**.

   The **Web chat** page opens, showing the settings for the web chat integration in the selected environment.

   Tip: The preview pane shows what the web chat will look like when embedded in a web page. If you see a message that starts with, `There is an error`, you probably haven't added any actions to your assistant yet. After you add an action, you can test the conversation from the preview pane.

3. Click the **Embed** tab.

   A code snippet is generated based on the web chat configuration. You (or a web developer) will add this code snippet to the web page where you want the web chat to appear.

   This code snippet contains an HTML `script` element. The script calls JavaScript code that is hosted on an IBM site and creates an instance of a widget that communicates with the assistant.

4. Click the **Copy to clipboard** icon to copy the embed script to the clipboard.
5. Edit the HTML source for the web page where you want the web chat widget to appear. Paste the code snippet into the page. Paste the code as close as possible to the closing `</body>` tag to ensure that your page renders faster.

   Important: Do not modify the `integrationID` or `region` property values in the generated embed script.

   If you aren't ready to add the web chat to a live website, you can quickly test it using a local HTML file. Use this HTML code as the source for a test page:

   ```
   <html>
   <head></head>
   <body>
      <title>My Test Page</title>
      <p>The body of my page.</p>
      &lt;!-- copied script elements --&gt;
      </body>
   </html>
   ```

   Copy this code into a file with the `.html` extension, and replace the `script` element with the embed script you copied in the previous step.

   Note: The identifiers in the embed script (such as `integrationID` `serviceInstanceID`) are not considered secret, and are visible to anyone who has access to your website. For more information, see Security.

6. If the system that hosts your website has limited Internet access (for example, if you use a proxy or firewall), make sure the URLs that host the web chat are accessible. For more information, see Access to web chat hosts.
7. Open the web page (or local test file) in your browser. You should see the launcher icon displayed on the page:

8. Click the launcher icon to open the chat window.
9. Paste the same embed script into each web page where you want the assistant to be available to your customers.

> **Tip:** You can paste the same script tag into as many pages on your website as you want. Add it anywhere where you want users to be able to reach your assistant for help. However, be sure to add it only once on each page.

You can now test your assistant and see its responses just as your customers would.

Before you go to production with the web chat, however, you will probably want to customize it for your site and for the needs of your customers. For more information, see Web chat development overview.

Adding the web chat to your mobile application

If you have a mobile application built on a mobile framework such as iOS, Android, or React Native, you can use a WebView with a JavaScript bridge to communicate between your app and the watsonx Assistant web chat.

Using WebViews with a JavaScript bridge is a common pattern with a similar implementation for all mobile frameworks.

Including the web chat as a WebView

You can include the web chat interface as part of a page of your mobile app, or as a separate panel that your app opens. In either case, you must host an HTML page that includes the web chat embed script, and then include that page as a WebView in your app.

In the embed script, use the `showLauncher` option to hide the web chat launcher icon, and the `openChatByDefault` option to open the web chat automatically when the page loads. In most cases, you will also want to use the `hideCloseButton` option and use the native controls of your app to control how the web chat page or panel closes. For more information about the configuration options you can specify in the embed script, see the Web chat API reference.

The following example shows an embed script that includes these configuration options:

```
$ <html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
  </head>
  <body>
    <script>
      window.watsonAssistantChatOptions = {
        // A UUID like '1d7e34d5-3952-4b86-90eb-7c7232b9b540' included in the embed code.
        integrationID: "YOUR_INTEGRATION_ID",
        // Your assistant's region, for example, 'us-south', 'us-east', 'jp-tok' 'au-syd', 'eu-gb', 'eu-de', and so on.
        region: "YOUR_REGION",
        // A UUID like '6435434b-b3e1-4f70-8eff-7149d43d938b' included in the embed code.
        serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",
        // The callback function that is called after the widget instance has been created.
        onLoad: function(instance) {
          instance.render();
        },
        showLauncher: false, // Hide the web chat launcher, you will open the WebView from your mobile application
        openChatByDefault: true, // When the web chat WebView is opened, the web chat will already be open and ready to go.
        hideCloseButton: true // And the web chat will not show a close button, instead relying on the controls to close the WebView
      };
      setTimeout(function(){const t=document.createElement('script');t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";document.head.appendChild(t);});
    </script>
  </body>
</html>
```

> **Note:** In your app, make sure you include logic to hide your web chat launching mechanism when the device is offline. If the device goes offline in the middle of a conversation, appropriate error messages and retries occur.

Using a JavaScript bridge

In some situations, your mobile application might need to communicate with the watsonx Assistant web chat. For example, you might need to set initial context data (such as a user ID or account information) or use the web chat security feature. To do this, you can use a JavaScript bridge.

A JavaScript bridge is a common pattern that can be used on all mobile platforms. Implementation details differ from one mobile application framework to

another; you can find specific examples on how to implement a JavaScript bridge for the framework you are using. However, the core concept of sending events back and forth between the mobile app and the web chat apply to all frameworks.

With a JavaScript bridge, events are sent between the mobile app and the WebView, and event listeners exist on both sides of the bridge to handle those events. Each event carries a message payload; because this payload is text-only, you must stringify and parse JSON objects to pass data back and forth.

If your mobile application needs to call a web chat instance method, you must call the method by sending an event from the app to the WebView using the JavaScript bridge. Similarly, if you need to run code in your mobile application in response to behavior in the web chat, you can send an event through the JavaScript bridge from the web chat to your mobile application.

You can make use of the `user_defined` response type and the `customResponse` event to drive behavior on your mobile application. However, you must strip the `event.data.element` property from the event before you pass it through the JavaScript bridge. Removing this property is necessary because it contains an HTML element, which cannot be stringified. Any use of the `user_defined` response type to write new views into the web chat must be written in HTML and JavaScript and handled inside the WebView. (For more information about the `customResponse` event, see the Web chat API reference.)

Managing user identity information in web chat

Your charges for watsonx Assistant are based on the number of unique monthly active users (MAU).

If you do not pass an identifier for the user when the session begins, the web chat creates one for you. It creates a first-party cookie with a generated anonymous ID. The cookie remains active for 45 days. If the same user returns to your site later in the month and chats with your assistant again, the web chat integration recognizes the user. And you are charged only once when the same anonymous user interacts with your assistant multiple times in a single month.

If you want to perform tasks where you need to know the user who submitted the input, then you must pass the user ID to the web chat integration. Choose a non-human-identifiable ID. For example, do not use a person's email address as the user ID.

In addition, the ability to delete any data created by someone who requests to be forgotten requires that a `customer_id` be associated with the user input. When a `user_id` is defined, the product can reuse it to pass a `customer_id` parameter. (For more information about deleting user data, see Labeling and deleting data.)

Because the `user_id` value that you submit is included in the `customer_id` value that is added to the `X-Watson-Metadata` header in each message request, the `user_id` syntax must meet the requirements for header fields as defined in RFC 7230.

To support these user-based capabilities, add the `updateUserID()` method in the code snippet before you paste it into your web page.

Note: If you enable security, you set the user ID in the JSON Web Token instead. For more information, see Authenticating users in web chat.

In the following example, the user ID `L12345` is added to the script.

```
<script>
  window.watsonAssistantChatOptions = {
    integrationID: 'YOUR_INTEGRATION_ID',
    region: 'YOUR_REGION',
    serviceInstanceID: 'YOUR_SERVICE_INSTANCE',
    onLoad: function(instance) {
      instance.updateUserID('L12345');
      instance.render();
    }
  };
  setTimeout(function(){
    const t=document.createElement('script');
    t.src='https://web-chat.global.assistant.dev.watson.appdomain.cloud/versions/' +
      (window.watsonAssistantChatOptions.clientVersion || 'latest') +
      '/WatsonAssistantChatEntry.js';
    document.head.appendChild(t);
  });
</script>
```

Apple devices

On Apple devices, the Intelligent Tracking Prevention feature automatically deletes any client-side cookie after 7 days. This means that if an anonymous customer accesses your website and then visits again two weeks later, the two visits are treated as two different MAUs.

To avoid this problem, use a server-side first-party cookie in your web application. For example, when an anonymous user visits your website for the first time, you can generate a unique user ID and store it in a server-side cookie with any expiration date you choose. Then, your code can use the `updateUserID()` instance method to set the user ID. You can then use the same cookie to set the same user ID for this customer on any future visits until it expires.

More information

For more information about billing, see User-based plans explained.

For more information about MAU limits per plan, see Pricing.

For more information about deleting a user's data, see Labeling and deleting data.

Supporting global audiences in web chat

You can build an assistant that understands customer messages in any of the languages that are supported by the service. The responses from your assistant are defined by you and can be written in any language you want.

However, some of the phrases that are displayed in the web chat widget are part of the web chat itself and do not come from the assistant. By default, these hardcoded phrases are specified in English, but you can apply a different language by adding lines to the embedded web chat script.

The hardcoded phrases used by the web chat widget are specified in *language pack* files. The web chat provides language packs that contain translations of each English-language phrase that is used by the web chat. You can instruct the web chat to use one of these other languages files by using the `instance.updateLanguagePack()` method.

Likewise, the web chat applies an American English locale to content that is added by the web chat unless you specify something else. The locale setting affects how values such as dates and times are formatted.

To configure the web chat for customers outside the US, follow these steps:

1. To apply the appropriate syntax to dates and times *and* to use a translation of the English-language phrases, set the locale. Use the `instance.updateLocale()` method.

   For example, if you apply the Spanish locale (`es`), the web chat uses Spanish-language phrases that are listed in the `es.json` file, and uses the `DD-MM-YYYY` format for dates instead of `MM-DD-YYYY` .

   > **Note:** The locale you specify for the web chat does not impact the syntax of dates and times that are returned by the underlying skill.

2. To change only the language of the hardcoded English phrases, use the `instance.updateLanguagePack()` method.

   For more information, see Instance methods.

3. By default, the text direction of the page is left to right. However, if you change the direction to "right to left", the webchat maintains your configuration.

Controlling the web chat version

The web chat JavaScript code follows semantic versioning practices. Starting with web chat version 7.0.0, you can set the version of the web chat that you want to use as a configuration option.

If you don't specify a version, the latest version is used automatically (`clientVersion: "latest"`). When you apply the latest version, you benefit from the continuous improvements, feature additions, and bug fixes that are made to the web chat regularly.

However, if you apply extensive customizations to your deployment, such as overriding the theme with your own custom theme, you might want to lock your deployment to a specific version. Locking on a version enables you to test new versions before you apply them to your live web chat.

To use a specific version (`clientVersion: "major.minor.patch"`), specify it as follows:

The following examples show what to specify when the current version is `7.7.1` .

- If you want to stay on a major version, but get the latest minor and patch releases, specify `clientVersion: "7"` .
- If you want to stay on a minor version, but get the latest patch releases, specify `clientVersion: "7.7.0"` .
- If you want to lock on to a specific minor version and patch release, specify `clientVersion: "7.7.1"` .

To test the updates in a version release of the web chat before you apply the version to your live web chat, follow these steps:

1. Lock the web chat that is running in your production environment to a specific version.
2. Embed your web chat deployment into a new page in a test environment, and then override the version lock setting. For example, specify `clientVersion: "latest"` .
3. Test the web chat, and make adjustments to the configuration if necessary.
4. Update your production deployment to use the latest version and apply any other configuration changes that you determined to be necessary after testing.

Here is an example embed script that locks the web chat version to `7.7.0`.

```
$ <html>
  <body>
    <script>
      window.watsonAssistantChatOptions = {
        integrationID: "YOUR_INTEGRATION_ID",
        region: "YOUR_REGION",
        serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",

        // This is what determines the version of web chat used. Note that this value is also used in the script src below.
        clientVersion: "7.7.0",

        onLoad: function(instance) {
          instance.render();
        },
      };
      setTimeout(function() {
        const t=document.createElement('script');
        t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" + (window.watsonAssistantChatOptions.clientVersion || 'latest') +
"/WatsonAssistantChatEntry.js";
        document.head.appendChild(t);
      });
    </script>
  </body>
</html>
```

For more information about features that were introduced in previous web chat versions, see the Web chat release notes.

Adding service desk support

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.

Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Integrating with Genesys

IBM Cloud

Integrate web chat with a Genesys Cloud service desk solution to provide help and personalize the customer service experience.

Connect Genesys Messenger to your AI-powered assistant with a web chat integration, and customize options as needed. If, in the course of a chat with your assistant, a customer requests contact with someone, you can transfer the conversation directly to a live agent.

Genesys allows your team to assist customers in real time, which increases customer satisfaction. And satisfied customers are happier customers. To learn more about this service desk solution, see the Genesys website.

Before you begin

You need:

- A Genesys Cloud license, as listed in **Prerequisites** at Getting started with web messaging
- An OAuth provider, only if allowing authenticated users (Optional)

Note: The watsonx Assistant web chat integration only supports Genesys Messenger, not the older Genesys web chat widget.

Setting up Genesys Messenger with your assistant

Web chat has a built-in integration to Genesys Web Messenger and currently only supports client-side configuration. It is not configured from the watsonx Assistant user interface (UI).

Configure Messenger

To define the appearance and behavior of the chat window:

1. Set up a web messenger by using the Genesys instructions at Configure Messenger.

   In **Messenger Configuration > Appearance**, select **Hide** for **Set your Launcher Button Visibility** and select **Off** for **User Interface**.

2. Set **Humanize your Conversation** to **On** if you want the name of the live agent displayed to users (Optional).

3. Enter the **name for your bot**, which is shown to users, in the **Bot Name** field, if you are using an architect flow to send messages before a live agent joins the conversation (Optional).

Deploy Messenger

To assign the configuration and generate a snippet to deploy the Messenger to your website, use the Genesys instructions at Deploy Messenger.

Enable the integration in web chat

1. Have ready the **script URL, environment, and deployment ID**, all values from the embed code that is provided by Genesys.

2. Use the script at Enabling the integration.

Your Genesys-powered service desk setup is complete, and customers can chat with your assistant and be connected to live agents.

Embed web chat inside Genesys Messenger

Live agents can better help customers and address their needs and concerns with details from the conversation history.

To embed web chat inside Genesys and give your live agents access to the conversation history, you need to set up an agent app, and then create and integrate with an interaction widget (Optional).

Configure the agent app

1. In the **OAuth settings** in Genesys, create a **new OAuth client**.

2. Set the **Grant Type** to **Token Implicit Grant**.

3. Add `https://web-chat.global.assistant.watson.appdomain.cloud/serviceDesks/genesysAgentApp.html?clientID=YOUR_CLIENT_ID&environment=YOUR_ENVIRONMENT` as a redirect URL.
4. Replace **YOUR_CLIENT_ID** with the **OAuth client ID**, and **YOUR_ENVIRONMENT** with the Genesys **environment value** from the embed code. Note that the client ID is unavailable when creating the OAuth client; you need to first create the client, and then add the redirect URL to it.

Create an interaction widget integration

1. Go to the **Integrations** page in Genesys, click the **+Integrations** button, select your integration, and then click **Install**.
2. Set the **Application URL** to `https://web-chat.global.assistant.watson.appdomain.cloud/serviceDesks/genesysAgentApp.html?clientID=YOUR_CLIENT_ID&environment=YOUR_ENVIRONMENT&conversationID={{pcConversationId}}`.
3. Replace **YOUR_CLIENT_ID** with the **OAuth client ID**, and **YOUR_ENVIRONMENT** with the Genesys **environment value** from the embed code.

> **Note:** The **conversationID={{pcConversationId}}** value is correct as is, and Genesys will replace **{{pcConversationId}}** with the current conversation ID when a live agent opens the widget.

4. Assign the integration to the queue that your agents use and to a group that agents belong to.

> **Note:** The widget will be available to all queues if you don't assign one; and the integration widget will be unavailable to agents if you don't assign a group.

Enable authenticated users

To configure your integration to allow only authenticated users to use Genesys Messenger, see Authenticated users.

Integrating with NICE CXone

IBM Cloud

Integrate web chat with a NICE CXone service desk solution and empower your live agents to provide faster solutions and better customer service.

Connect NICE CXone to your AI-powered assistant with a web chat integration, and personalize the experience as necessary. If, in the course of conversation with your assistant, a customer requests contact with someone, you can transfer or escalate directly to a live agent.

NICE CXone allows your team to assist customers in real time, which increases customer satisfaction. And satisfied customers are happier customers. To learn more about this service desk solution, see the NICE website.

Before you begin

You need to sign up for a NICE CXone account, or have an existing one, and contact the NICE CXone team to access and use Digital First Omnichannel (DFO) Live Chat.

An OAuth provider is required, if only allowing authenticated users.

> **Note:** The watsonx Assistant web chat integration only supports NICE CXone DFO Live Chat, not the older NICE CXone chat channel integration.

Setting up NICE CXone with your assistant

Web chat has a built-in integration to the NICE CXone DFO live messaging channel and currently only supports client-side configuration. It is not configurable from the watsonx Assistant user interface (UI).

Configure DFO Live Chat

To set up one or more live chat channels to integrate with your assistant, see NICE CXone instructions at Set Up Digital First Omnichannel Live Chat.

Enable the integration in web chat

1. Have ready the **DFO channel ID, brand ID, and environment**, all values found in the embed code provided by NICE CXone.
2. Use the script at Enabling the integration.

Enable authenticated users

To configure your integration to only allow authenticated users, see Authenticated users.

Integrating with Salesforce

IBM Cloud

Integrate your web chat with a Salesforce service desk solution so your customers always get the help they need.

Integrate with a Salesforce service desk by deploying your assistant with the web chat integration. The web chat serves as the client interface for your assistant. If, in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Salesforce agent.

Salesforce is a customer relationship management solution that brings companies and customers together. It is an integrated CRM platform that gives all your departments, including marketing, sales, commerce, and service, a single and shared view of every customer.

Before you begin

To connect to a Salesforce service desk, your organization must have a Salesforce Service Cloud plan that supports Live Agent Chat. Chat support is available in Salesforce Service Cloud Unlimited and Enterprise plans. It is also available with Performance or Developer plans that were created after 14 June 2012.

Your organization must have a Salesforce chat app with the following characteristics:

- Console navigation
- Navigation items: Cases, Chat sessions, Chat transcripts
- User profiles: Apply the appropriate profiles to ensure that agents can access the app and view chat history information. You can limit access to this page later. See Profiles.
- A chat deployment.
- A chat button deployment.
- Routing must be configured for the chat button. See Chat routing options.
- If you choose omni-channel routing, be sure to include omni-channel as a utility in the chat app. See Omni-Channel.

You must have a level of access to your Salesforce service desk deployment that allows you to do the following:

- Edit the chat app
- Get chat deployment and button code details
- Add custom fields to layout objects
- Create Visualforce pages

If you don't, ask someone with the appropriate level of access to perform this procedure for you.

Setting up the Salesforce service desk connection

To set up a Salesforce service desk integration, complete the following steps:

1. Go to your web chat settings. For more information, see Integrating the web chat with your website.
2. From the web chat integration page in watsonx Assistant, set the **Allow transfers to live agents** switch to **On**, and then choose **Salesforce** as the service desk type. Click **Next**.
3. For watsonx Assistant to connect to a Salesforce service desk, it needs information about your organization's Salesforce chat deployment and button implementations. Specifically, it needs the API endpoint, organization ID, deployment ID, and button ID. The service can derive the values that it needs from code snippets that you copy and paste to this configuration page.

   Tip: In a separate browser tab or window, open your Salesforce account settings page. Log in with a user ID that has administrative privileges. You must switch back and forth between your Salesforce and watsonx Assistant web chat integration setup pages. It's easier to do so if you have both pages open at once.

   - Get the deployment code for your Salesforce Agent Configuration chat deployment.

     Go to the Salesforce **Feature Settings > Service > Chat > Deployments** page. Find your organization's deployment. Scroll to the end of the chat deployment configuration page and copy the *Deployment Code* snippet.

   - Paste the deployment code snippet into the **Deployment code** field in the watsonx Assistant Salesforce configuration page.
   - Get the Chat Button code.

     Go to the Salesforce **Feature Settings > Service > Chat > Chat Buttons & Invitations** page. Find your organization's button implementation.

Scroll to the end of the page, and then copy the *Chat Button Code* snippet.

- Paste the chat button code snippet into the **Chat button code** field in the watsonx Assistant Salesforce configuration page, and then click **Next**.

4. Add a chat app that enables the Salesforce agent to see a history of the chat. To do so, create a Visualforce page, and then add a chat app to the page.

5. Add custom fields to the Salesforce chat transcript layout.

   Note: This is a one-time task. If the fields already exist for your organization, you can skip this step.

   These custom fields are referenced from the Visualforce page code that you will use in the next step.

   See Create Custom Fields.

   From the Salesforce **Data >Objects and Fields > Object Manager > Chat Transcript > Fields & Relationships** page, create the following custom fields:

   - **Token**: Stores a watsonx Assistant authentication token that secures the communication between Salesforce and your assistant.
     - **Data Type**: Text Area
     - **Field Label**: `x-watson-assistant-key`

6. Create a Visualforce page.

   Visualforce pages are the mechanism that Salesforce provides for you to customize a live agent's console by adding your own pages to it. A Visualforce page is similar to a standard web page, but it provides ways for you to access, display, and update your organization's data. Pages can be referenced and invoked by using a unique URL, just as HTML pages on a traditional web server can be. See Create Visualforce Pages

   - From the web chat integration page in watsonx Assistant, copy the code snippet from the Visualforce page markup field.
   - Switch to your Salesforce web page. Search for **Visualforce Pages**. Create a page. Add a label and name to the page. Select the *Available for Lightning Experience, Lightning Communities, and the mobile app* checkbox. Paste the code snippet that you copied in the previous step into the page markup field.

7. Add the Visualforce page that you created to the Salesforce chat app.

   To ensure the Salesforce agents can see history of the chat between the customer and your assistant, you must add the page that you created earlier into the console that they use to keep track of their work. See Create and Configure Lightning Experience Record Pages.

   - From the Salesforce App Launcher, open the chat app that you created for your agents to talk to customers.
   - Open the *Chat Transcripts* object, and then select a transcript page.
   - Click the *Setup* icon, and then select *Edit Page*.
   - Drag the Visualforce component and drop it into the Chat Transcript Record page layout where you want the chat window to be displayed.
   - In the component editor, select the Visualforce page that you created earlier, make any adjustments to the component height that you want, and then click *Save*.

     Important: If you do make changes, make sure the height of the Visualforce page is 20 px smaller than the height of the component that you add it to. By default, the height of the component is 300 px and the height of the Visualforce page is 280 px. (The height of the Visualforce page is specified in the `height` attribute of the `iframe` HTML element in the code snippet that you copy and paste.)

   - Click *Activation*, and then click the APP, RECORD TYPE, AND PROFILE tab.
   - Select the apps to which you want to apply the page layout, and then click *Next*.
   - Select the appropriate record type, such as Main, and then click *Next*.
   - Select user profiles to give the appropriate set of users access to the page. Limit the group to include only those who you want to be able to view chat history information in the page.
   - Click *Next*, and then click *Save*.

8. From the Salesforce configuration page in watsonx Assistant, click **Save and exit** to finish setting up the connection.

When you test the service desk integration, make sure there is at least one agent with `Available` status.

Watch a 5-minute video that provides an overview of setting up a connection to a Salesforce service desk:

View video: Salesforce Integration: watsonx Assistant

Adding transfer support to your actions

Update an action to make sure it understands when users request to speak to a person, and can transfer the conversation properly.

Routing based on browser information

When a customer interacts with the web chat, information about the current web browser session is collected. For example, the URL of the current page is collected. You can use this information to add custom routing rules to your actions. For example, if the customer is on the Products page when a transfer to a human is requested, you might want to route the chat transfer to agents who are experts in your product portfolio. If the customer is on the Returns page, you might want to route the chat transfer to agents who know how to help customers return merchandise.

Integrating with Zendesk

IBM Cloud

Integrate web chat with a Zendesk service desk solution, so your customers always get the help they need.

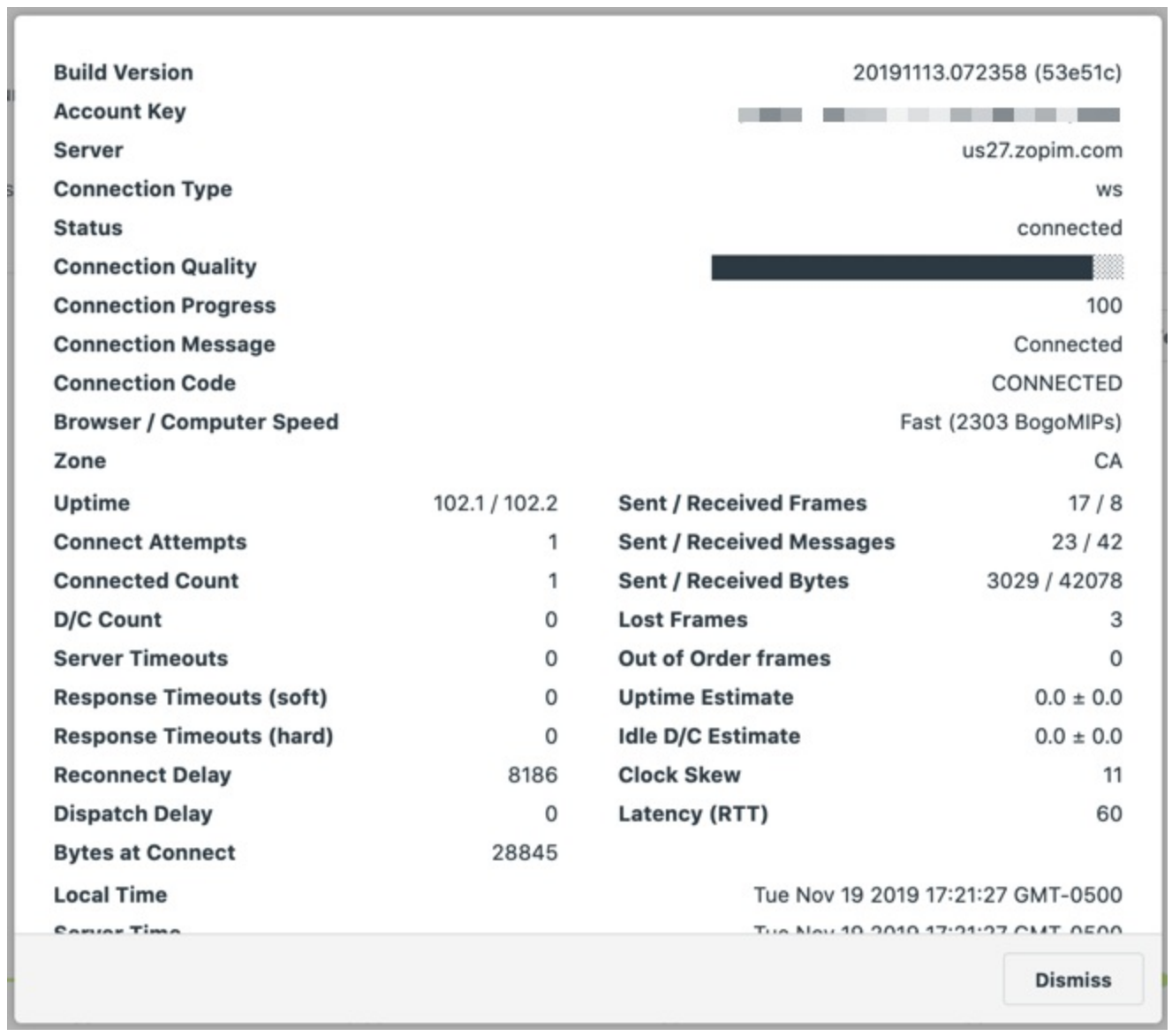
Connect to Zendesk by deploying your assistant with the web chat integration that serves as the client interface. If, in the course of a chat with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a live agent.

Zendesk allows you to assist customers in real time, which increases customer satisfaction. And satisfied customers are happier customers. To learn more about this service desk solution, see the Zendesk website.

Before you begin

⚠ **Important:** Zendesk is updating the Zopim Chat REST API domain. If you use a previous version of the Web Chat that integrates with Zendesk, you must upgrade to the latest version as soon as possible. For more information about how to manage your web chat version, see Controlling the web chat version.

1. Sign up for or switch to a Zendesk Service account with an Enterprise plan, which is required.
2. Create a subdomain for your service desk. After you sign up and specify a subdomain, your Zendesk console is available from a URL with the syntax: `<subdomain>.zendesk.com`. For example, `ibm.zendesk.com`.
3. Log in to your Zendesk subdomain.
4. Open the Zendesk **Dashboard**.
5. Click the **Products** icon (four blocks) in the header, and then select the **Chat** icon.
6. Click your profile, and then select **Check Connection**.
7. Keep this screen open for the **Connect Zendesk to your assistant** step.



You also need to decide whether to enable security for Zendesk after setup. More information at Securing the transfer to Zendesk.

Setting up the Zendesk service desk connection

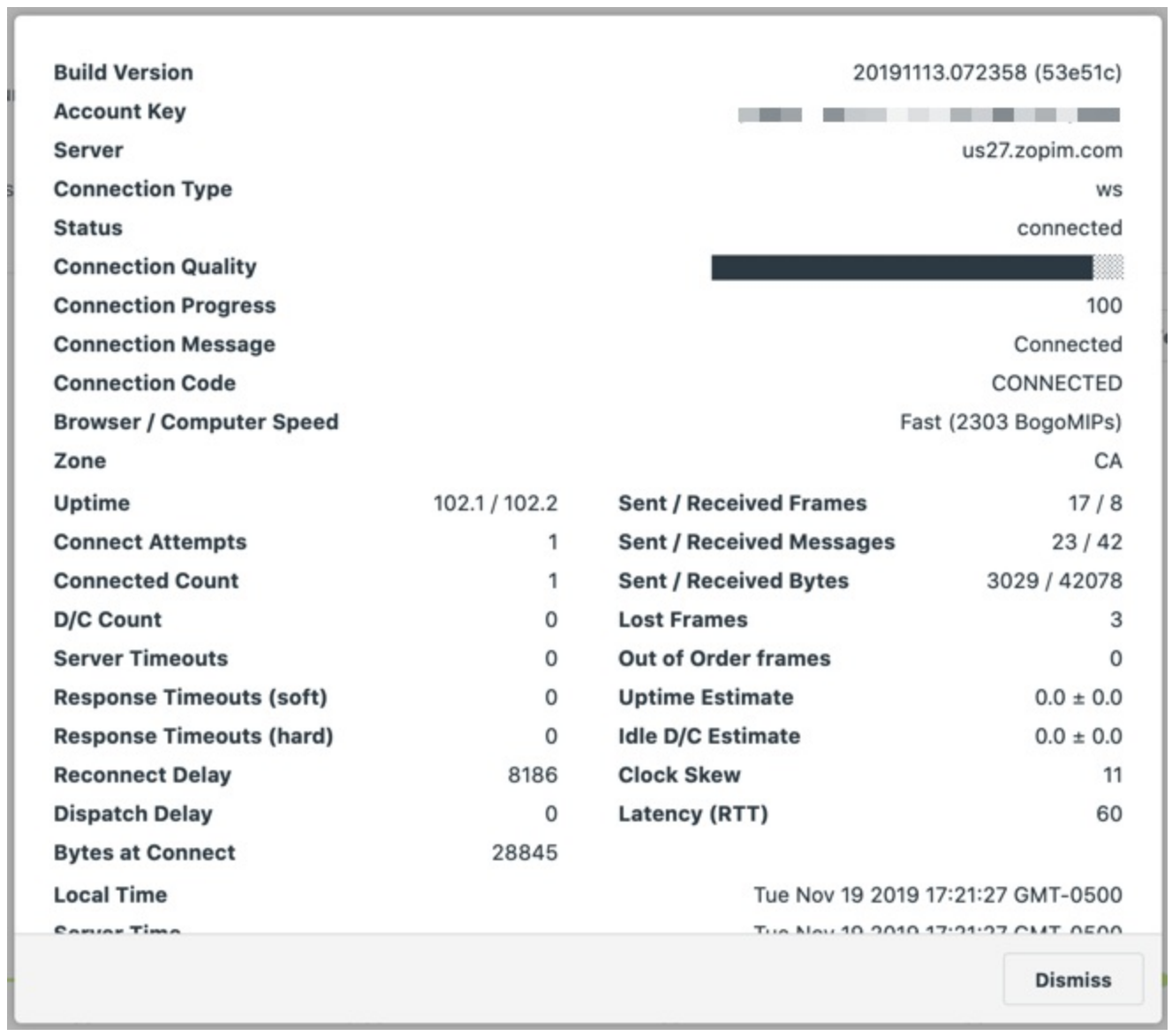
In your watsonx Assistant installation:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu. For more information, see Integrating the web chat with your website.
2. Click **Web chat** and then click **Open**.
3. Select an environment, and click **Confirm**.
4. Go to the **Live agent** tab.
5. Click **Zendesk**.

Connect Zendesk to your assistant

In the **Live agent** tab of your Zendesk web chat integration:

1. Toggle **Transfers enabled**. Two tasks should be displayed: **Connect Zendesk** and **Private app Install**.
2. Copy the **Account key** from your Zendesk account in the tab that you kept open.



3. Paste into the **Account key** field in your assistant.
4. Click **Connect account**. A checkmark indicates successful completion.

Install app in Zendesk

IBM provides an application to install in your Zendesk Service subdomain. When a customer asks to speak to someone, your assistant shares a chat summary for the transferred conversation with the Zendesk agent by using this private app.

1. Click **Private app Install**.
2. Click the arrow next to **Download the watsonx Assistant Zendesk app**. The app appears in the location of your downloads.

   Note: On Safari, application files are extracted from the ZIP file into a folder. To keep the file archived as a .zip file so you can upload it later, edit your Safari preferences. Clear the **Open safe files after downloading** checkbox.

3. In your Zendesk installation, click the **Products icon** and go to the **Admin Center**.
4. Click **Apps and integrations**.
5. Select **Zendesk Support apps**.
6. Click **Upload private app**.
7. Click **Choose File**, select the app that you downloaded from your watsonx Assistant installation, and click **Upload**.
8. Click **Install**, if you agree to the Marketplace Terms of Use that display in the **Creating a new App** window.
9. Enter a name for your app in the **Title** field, and click **Install**.

The app is now listed at **My Apps** under **Private apps**, and can be enabled, disabled, or deleted. For more information, see Uploading and installing a private app in Zendesk.

When you test the service desk integration, ensure that there is at least one agent with `Online` status. Agent status is set to `Invisible` unless it is explicitly changed.

Turn on Agent Workspace

Zendesk Agent Workspace brings Zendesk Chat and Zendesk Support together, so all your customer interactions are in one place, and communication is seamless, personal, and efficient. That means more productive agents and happy customers.

In Zendesk:

1. Click the **Products icon** and go to the **Admin Center**.
2. Click **Workspaces**.
3. Click the **Turn on Agent Workspace** button. The green **On** box displays.

Agent Workspace should now feature on several screens in Zendesk Support, including on the **Dashboard** with tickets, the **Visitors** page, and in the menu as **Conversations** where agents can accept chats from customers waiting for assistance.

Securing the transfer to Zendesk

You must collect the name and email address of each user if enabling security in Zendesk. This information must be passed to the web chat so it can be provided to Zendesk when the conversation is transferred.

When you add security to your Zendesk integration, you ensure that the visitors you are helping are legitimate customers. Enabling visitor authentication also enables support for cross-domain traffic and cross-browser identification. For more information, see the Enabling authenticated visitors in the Zendesk chat widget.

Before you can secure the Zendesk connection, complete the following required tasks:

1. Secure the web chat. For more information, see Securing the web chat.
2. Encrypt sensitive information that you pass to the web chat.

When you enable security in Zendesk, you must provide the name and email address of the current user with each request. Configure the web chat to pass this information in the payload.

Specify the information by using the following syntax. Use the exact names (`name` and `email`) for the two name and value pairs.

```
{
user_payload : {
        name: '#{customerName}',
        email: '#{customerEmail}'
  }
}
```

For more information, see Encrypting sensitive data in web chat.

Zendesk also expects `iat` and `external_id` name and value pairs. However, you need not provide this information. IBM automatically provides a JWT that contains these values.

For example:

```
const userPayload = {
 "name" : "Cade Jones",
 "email" : "cade@example.com",
}
```

```
// Sample NodeJS code on your server.
const jwt = require('jsonwebtoken');
const RSA = require('node-rsa');

const rsaKey = new RSA(process.env.PUBLIC_IBM_RSA_KEY);

/**
* Returns a signed JWT. Optionally, adds an encrypted user_payload in stringified JSON.
*/
function mockLogin(userID, userPayload) {
   const payload = {
     sub: userID, // Required
     iss: 'www.ibm.com', // Required
     acr: 'loa1' // Required
     // A short-lived exp claim is automatically added by the jsonwebtoken library.
   };
   if (userPayload) {
       // If there is a user payload, it is encrypted in base64 format using the IBM public key.
       payload.user_payload = rsaKey.encrypt(userPayload, 'base64');
   }
   const token = jwt.sign(payload, process.env.YOUR_PRIVATE_RSA_KEY, { algorithm: 'RS256', expiresIn: '10000ms' });
   return token;
```

```
}
```

3. From the Zendesk application, enable visitor authentication.
   - From the Chat dashboard navigation pane, expand *Settings*, and then click *Widget*.
   - Open the *Widget security* tab.
   - In the *Visitor Authentication* section, click the *Generate* button.

   For more information, see Enabling authenticated visitors in the Zendesk chat widget. You do not need to follow the steps to create a JWT. The Assistant service generates a JSON Web Token for you.
4. Copy the shared secret from Zendesk.

To secure the Zendesk connection, complete the following steps:

1. In the *Authenticate users* section, set the switch to **On**.
2. Paste the secret that you copied from the Zendesk setup page into the **Zendesk shared secret** field.
3. Decide whether to allow unidentified users to access Zendesk.

   The web chat integration allows anonymous users to initiate chats. However, as soon as you enable visitor authentication, Zendesk requires that the name and email of each user be provided. If you try to connect without passing the required information, a connection is refused.

   If you want to allow anonymous users to connect to Zendesk, you can provide fictitious name and email data. Write a function to populate the two fields with fictitious name and email values.

   For example, your function must check whether you know the name and email of the current user, and if not, add canned values for them:

```
const userPayload = {
 "name" : "Jane Doe1",
 "email" : "jdoe1@example.com",
}
```

   After you write a function to ensure that name and email values are always provided, set the *Authenticate anonymous user chat transfers* switch to **On**.

Note: For security reasons, the `secret` authentication fields are removed from view after initial setup.

Adding transfer support to your actions

Update an action to make sure it understands when users request to speak to a person, and can transfer the conversation properly.

Routing based on browser information

When a customer interacts with the web chat, information about the current web browser session is collected. For example, the URL of the current page is collected.

You can use this information to add custom routing rules to your actions. For example, if the customer is on the Products page when a transfer to a human is requested, you can route the chat to agents who are experts in your product portfolio.

If the customer is on the Returns page, you might want to route the chat transfer to agents who know how to help customers return merchandise.

Tutorials

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.

Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

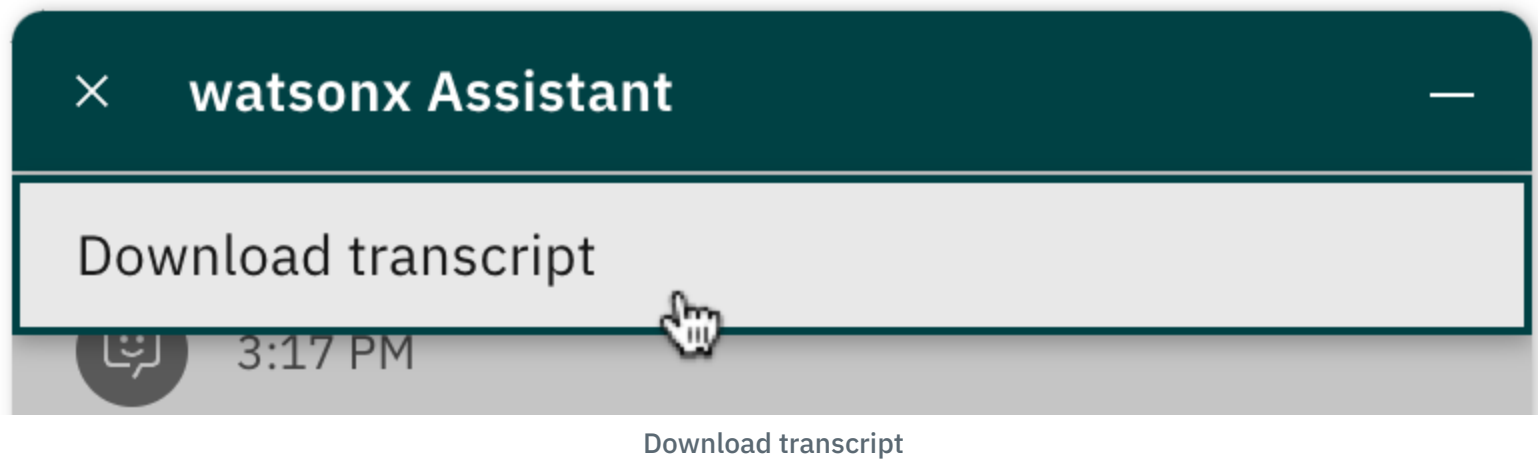
Administering your instance

Tutorial: Providing a downloadable conversation transcript

You can customize the web chat to offer your customers the option of downloading a transcript of the conversation history.

Note: For a complete, working version of the example that is described in this tutorial, see Download history for watsonx Assistant web chat .

To support downloading a conversation transcript, this example adds a custom menu option to the overflow menu in the header of the chat window:



Download transcript

Clicking this menu option initiates downloading of a file that contains the complete conversation history in comma-separated values (CSV) format.

1. Create a handler for the `send` and `receive` events. In this handler, save each incoming or outgoing message in a list (`messages`) to maintain a history of the conversation.

```
$ const messages = [];

function saveMessage(event) {
  messages.push(event.data);
}
```

2. Create a handler for the `history:begin` event, which is fired when the web chat is reloaded from the session history. In this handler, save any reloaded session history to the list.

```
$ function saveHistory(event) {
  messages.push(...event.messages);
}
```

3. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `send`, `receive`, and `history:begin` events, registering the appropriate handlers as callbacks.

```
$ instance.on({ type: 'send', handler: saveMessage });
instance.on({ type: 'receive', handler: saveMessage });
instance.on({ type: 'history:begin', handler: saveHistory });
```

4. Create a function that converts the messages that are saved in the `messages` list to the format you want to provide in the downloaded file. This conversion needs to accommodate any response types that the conversation might include (such as text, images, options, or transfers to a human agent).

   In this example, we convert the messages into a CSV file format that can be opened with an application such as Microsoft Excel. The first column in each line is a label that indicates whether the message originated from the customer (`You`) or from the assistant (`Lendyr`).

   Note: This function relies on a helper function (`createDownloadText`) that formats the text for each line. You can see the implementation of this helper function in the full example.

```
$ function createDownload() {
  const downloadLines = [createDownloadText('From', 'Message')];

  messages.forEach(message => {
    if (message.input?.text) {
      // This is a message that came from the user.
      downloadLines.push(createDownloadText('You', message.input.text));
    } else if (message.output?.generic?.length) {
      // This is a message that came from the assistant. It can contain an array of individual message items.
      message.output?.generic.forEach(messageItem => {
        // This is only handling a text response but you can handle other types of responses here as well as
        // custom responses.
        if (messageItem?.text) {
          downloadLines.push(createDownloadText('Lendyr', messageItem.text));
        }
      });
    }
  });

  return downloadLines.join('\n');
}
```

5. Create a function that initiates the download of the conversation history file. This function calls the `createDownload()` function to generate the content to download. It then simulates clicking a link to start the download, by using a file name generated from the current date.

```
$ function doDownload() {
  const downloadContent = createDownload();

  const blob = new Blob([downloadContent], { type: 'text/csv' });
  const url = URL.createObjectURL(blob);

  // To automatically trigger a download, we have to create a fake "a" element and then click it.
  const timestamp = new Date().toISOString().replace(/[_:]/g, '-').replace(/.[0-9][0-9][0-9]Z/, '');
  const a = document.createElement('a');
  a.setAttribute('href', url);
  a.setAttribute('download', `Chat History ${timestamp}.csv`);
  a.click();
}
```

6. In your `onLoad` event handler, use the `updateCustomMenuOptions()` instance method to add a custom menu option that customers can use to download the conversation history. Add this line immediately before the call to the `render()` instance method.

```
$ instance.updateCustomMenuOptions('bot', [{ text: 'Download history', handler: doDownload }]);
```

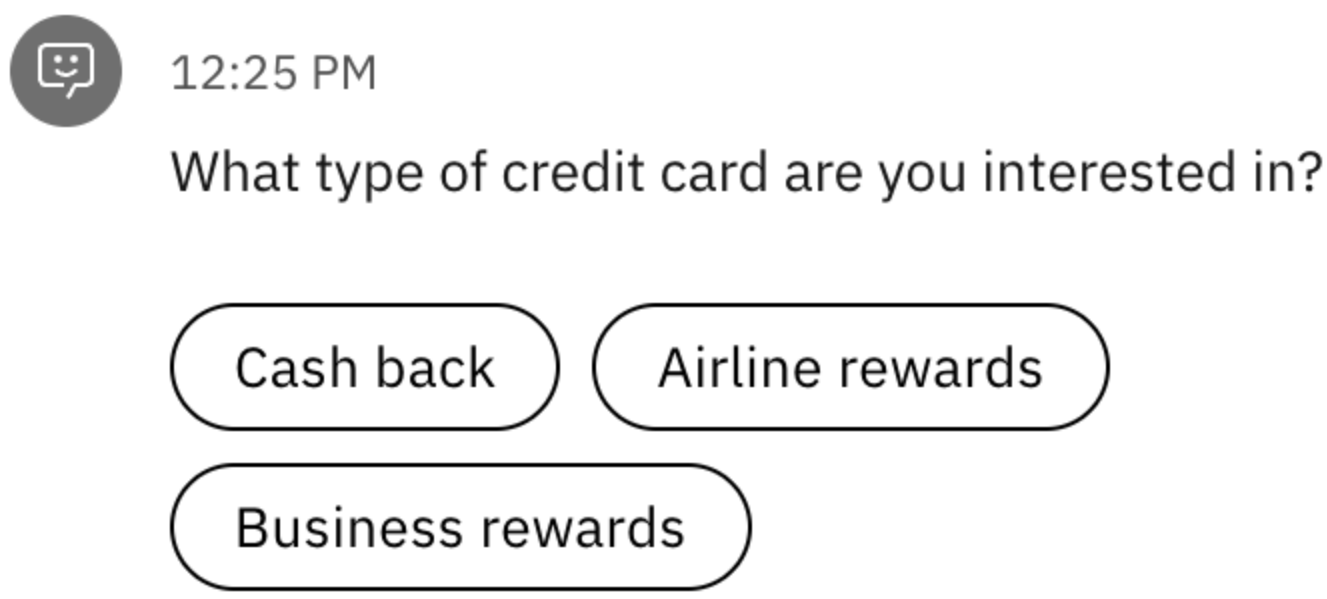
For complete working code, see the Download history for watsonx Assistant web chat example.

Tutorial: implementing custom option buttons in the web chat

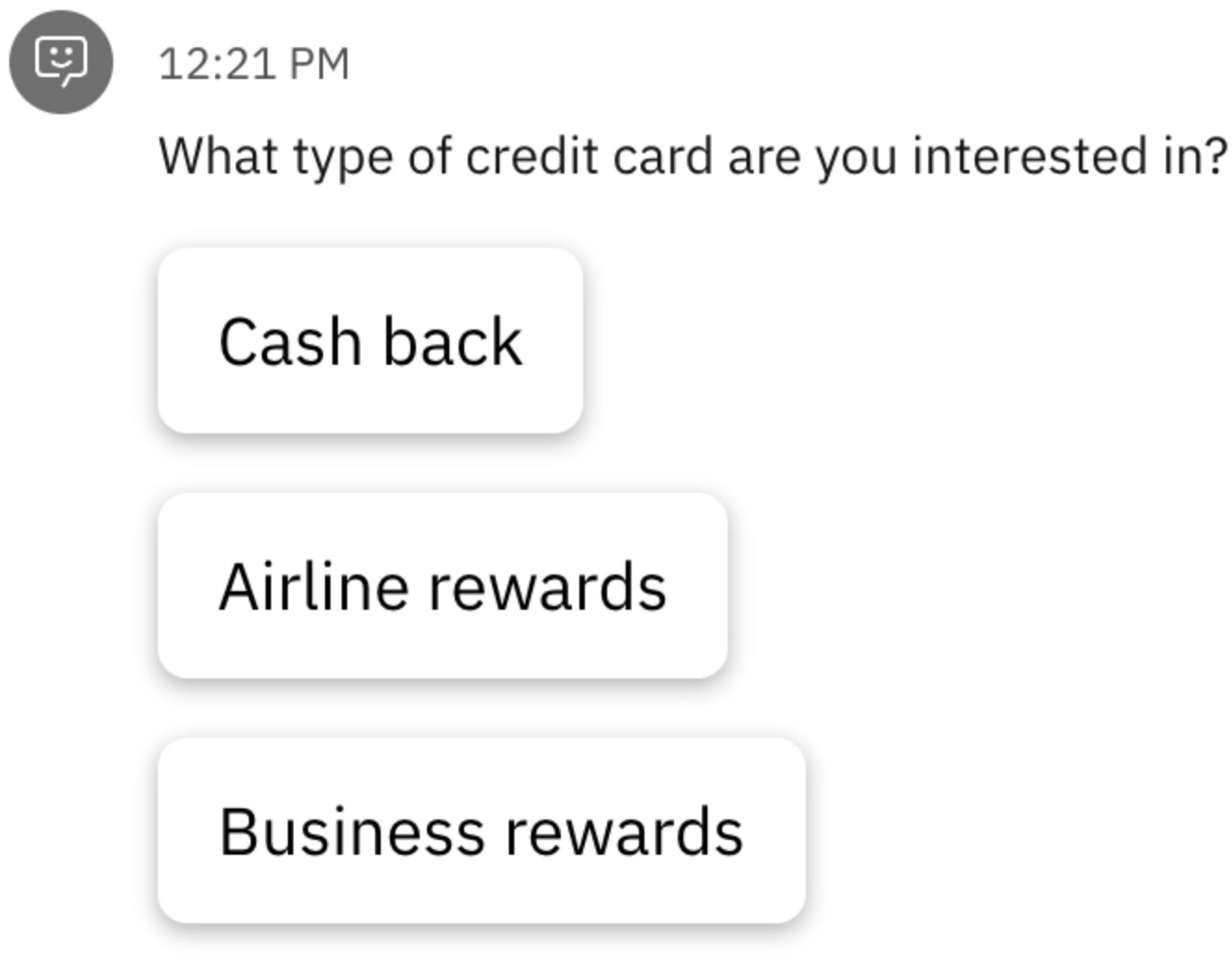
This tutorial shows how you might replace the default rendering of an options response with your own custom clickable buttons.

Note: For a complete, working version of the example described in this tutorial, see Custom buttons for watsonx Assistant web chat.

By default, the web chat always displays an options response as a set of clickable buttons (for 4 or fewer options) or as a drop-down list (for 5 or more more options). This example shows the default rendering of an options response with 3 options:



For this tutorial, we will replace this default rendering with larger, card-style buttons:



Because the rendering of an options response cannot be modified, we will do this by intercepting any incoming options responses from the assistant and converting them into custom (`user_defined`) responses. We can then implement a custom rendering for these responses.

1. Create a handler for the `pre:receive` event. In this handler, look for any `option` responses in the message payload, and convert them into `user_defined` responses.

```
$ function preReceiveHandler(event) {
  const message = event.data;
  if (message.output.generic) {
    message.output.generic.forEach(messageItem => {
      if (messageItem.response_type === 'option') {
        messageItem.response_type = 'user_defined';
      }
    })
  }
}
```

2. Create a handler for the `customResponse` event. This handler renders the custom buttons, using a custom `CardButton` style we can define in the CSS. (You can see the definition of this style in the full example.)

```
$ function customResponseHandler(event) {
  const { message, element, fullMessage } = event.data;
  message.options.forEach((messageItem, index) => {
    const button = document.createElement('button');
    button.innerHTML = messageItem.label;
    button.classList.add('CardButton');
    button.addEventListener('click', () => onClick(messageItem, button,     fullMessage, index));
    element.appendChild(button);
  });
}
```

3. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `pre:receive` and `customResponse` events, registering the handlers as callbacks.

```
$ instance.on({ type: 'customResponse', handler: customResponseHandler });
instance.on({ type: 'pre:receive', handler: preReceiveHandler });
```

4. Create a click handler to respond when the customer clicks on one of the custom buttons. In the handler, use the `send()` instance method to send a message to the assistant, using the button label as the message text.

   In addition, we're adding the custom CSS class `CardButton--selected` to the clicked button, changing its appearance to show that it was selected. (This class is also defined in the full example.)

```
$ function onClick(messageItem, button, fullMessage, itemIndex) {
  webChatInstance.send({ input: { text: messageItem.label }});
  button.classList.add('CardButton--selected');
}
```

5. If the user reloads the page or navigates to a different page, the web chat reloads from the session history. If this happens, we want to preserve the "selected" state of any clicked buttons.

   To do this, in the `onClick` handler, use the `updateHistoryUserDefined` instance method to store a variable in the session history that indicates which button was clicked.

```
$ webChatInstance.updateHistoryUserDefined(fullMessage.id, { selectedIndex:     itemIndex });
```

   Then, in the `customResponse` handler, read this value and use it to set the initial states of the buttons in any custom responses already in the session history.

```
$ if (fullMessage.history?.user_defined?.selectedIndex === index) {
  button.classList.add('CardButton--selected');
}
```

For complete working code, see the Custom buttons for watsonx Assistant web chat example. The example also shows how to disable the buttons in a custom response after the customer has sent a message, which prevents using the buttons to send a message out of order.

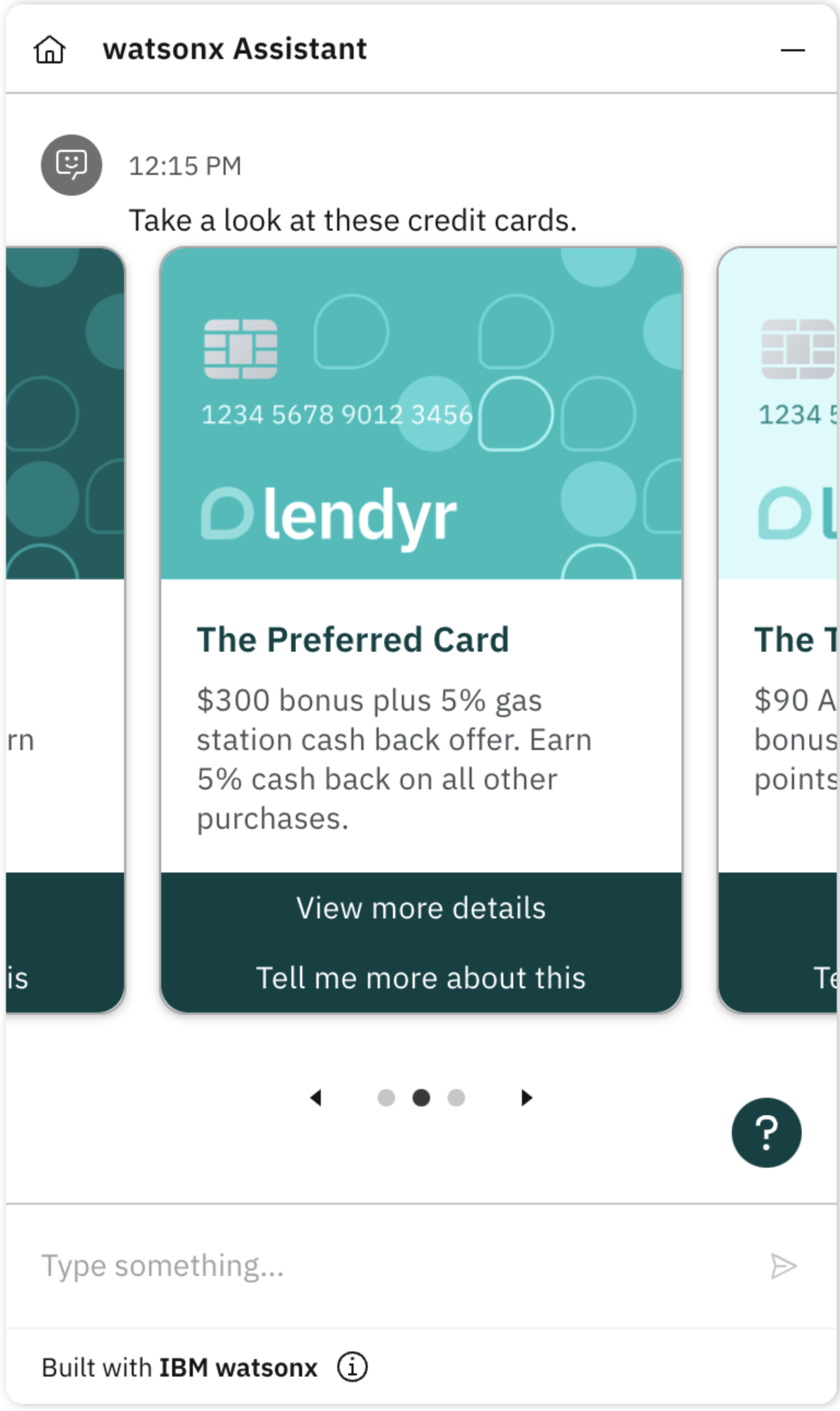
Tutorial: Rendering a custom response as a content carousel

This tutorial shows how you might use custom responses to render information in the form of a content carousel.

Note: For a complete, working version of the example that is described in this tutorial, see Content carousel for watsonx Assistant web chat.

A *content carousel* (or *slider*) is a type of interactive element that shows options as a scrollable series of slides.

watsonx Assistant does not have a built-in response type for content carousels. Instead, you can use the `user_defined` response type to send a custom response with the information you want to show, and extend the web chat to render the content carousel by using standard JavaScript libraries.



Content carousel

This example shows how you can use the Swiper library to render a custom response as a content carousel.

1. In the action step that you want to create a content carousel, use the JSON editor to define a `user_defined` custom response, which can contain any data that you want to include. Inside the response, specify the data that is required for populating the content carousel. In this example, we're sending information about various types of credit cards, which we display in a custom response:

```
$ {
  "generic": [
    {
      "user_defined": {
        "carousel_data": [
          {
            "alt": "everyday card",
            "url": "lendyr-everyday-card.jpg",
            "title": "The Everyday Card",
            "description": "$300 bonus plus 5% gas station cash back offer. Earn 2% cash back on all other purchases."
          },
          {
            "alt": "preferred card",
            "url": "lendyr-preferred-card.jpg",
            "title": "The Preferred Card",
            "description": "$300 bonus plus 5% gas station cash back offer. Earn 5% cash back on all other purchases."
          },
          {
            "alt": "topaz card",
            "url": "lendyr-topaz-card.jpg",
            "title": "The Topaz Card",
            "description": "$90 Annual fee. Earn 120,000 bonus points. Earn additional points on every purchase."
          }
        ],
        "user_defined_type": "carousel"
      },
      "response_type": "user_defined"
    }
  ]
}
```

Note: In this example, we are including the carousel data inside the `user_defined` response. Depending on the design of your assistant, another option would be to store the data in skill variables that can be accessed by web chat from the `context` object.

2. Create a handler for the `customResponse` event. This handler renders the content carousel, by using the styles defined by the Swiper library. (You can see the definitions of these styles in the full example.)

   This function also relies on a helper function (`createSlides()`), which we will create in the next step. The complete code for this function also initializes the Swiper library. For more information, see the full example.

```
$ function carouselCustomResponseHandler(event, instance) {
  const { element, message } = event.data;

  element.innerHTML = `
    <div class="Carousel">
      <div class="swiper">
        <div class="swiper-wrapper"></div>
      </div>
      <div class="Carousel__Navigation" >
        <button type="button" class="Carousel__NavigationButton Carousel__NavigationPrevious bx--btn bx--btn--ghost">
          <svg fill="currentColor" width="16" height="16" viewBox="0 0 32 32" aria-hidden="true"><path d="M20 24L10 16 20 8z"></path></svg>
        </button>
        <div class="Carousel__BulletContainer"></div>
        <button type="button" class="Carousel__NavigationButton Carousel__NavigationNext bx--btn bx--btn--ghost">
          <svg fill="currentColor" width="16" height="16" viewBox="0 0 32 32" aria-hidden="true"><path d="M12 8L22 16 12 24z"></path></svg>
        </button>
      </div>
    </div>`;

  // Once we have the main HTML, create each of the individual slides that will appear in the carousel.
  const slidesContainer = element.querySelector('.swiper-wrapper');
  createSlides(slidesContainer, message, instance);

...

}
```

3. Create the `createSlides()` helper function that renders each slide in the content carousel. This function retrieves the values from the custom response

(`alt`, `url`, `title`, and `description`), and uses them to populate the HTML for the slide. (For the complete function, see the full example.)

```
$ carouselData.forEach((cardData) => {
  const { url, title, description, alt } = cardData;
  const cardElement = document.createElement('div');
  cardElement.classList.add('swiper-slide');

  cardElement.innerHTML = `
    <div class="bx--tile Carousel__Card">
      <img class="Carousel__CardImage" src="${url}" alt="${alt}" />
      <div class="Carousel__CardText">
        <div class="Carousel__CardTitle">${title}</div>
        <div class="Carousel__CardDescription">${description}</div>
      </div>

      <a href="https://www.ibm.com" class="Carousel__CardButton bx--btn bx--btn--primary" target="_blank">
        View more details
      </a>

      <button type="button" class="Carousel__CardButton Carousel__CardButtonMessage bx--btn bx--btn--primary">
        Tell me more about this
      </button>
    </div>
  `;

  ...

});
```

4. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `customResponse` event, registering the `carouselCustomResponseHandler()` function as the callback.

```
$   instance.on({
    type: 'customResponse',
    handler: (event, instance) => {
      if (event.data.message.user_defined && event.data.message.user_defined.user_defined_type === 'carousel') {
        carouselCustomResponseHandler(event, instance);
      }
    },
  });
```

> **Note:** In this example, we are checking the custom `user_defined_type` property of the custom response, and calling the `carouselCustomResponseHandler()` function only if the specified type is `carousel`. This optional check shows how you might use a custom property to define multiple different custom responses (each with a different value for `user_defined_type`).

For complete working code, see the Content carousel for watsonx Assistant web chat example.

Tutorial: Setting context variables for actions from the web chat

This tutorial shows how you can use the web chat to set the values of context variables that your actions can access.

> **Note:** For a complete, working version of the example described in this tutorial, see Setting context variables for watsonx Assistant web chat.

You can use context variables to store information that the assistant uses throughout the conversation.

Context data is maintained throughout the session. The context is sent to the assistant as part of each message, and returned with each response, in an object called `context`. Context variables for actions skills are stored in the following location:

```
$ "context": {
  "skills": {
    "action skill": {
      "skill_variables": {
        ...
      }
    }
  }
```

```
}
```

Any JSON data can be stored in the `skill_variables` object and read or modified by either the actions or the web chat.

This example shows how you can store the customer's name in a context variable, which you can then use to show a personalized greeting. You can use this same approach to store any other information that your assistant might use. Examples might include the customer's location, an account balance, or stored preferences.

To set the value of a context variable from the web chat, follow these steps:

1. Create a handler for the `pre:send` event. This handler modifies the payload of outgoing message events to assign a value to a context variable called `User_Name`.

   We want to set the user name only once at the beginning of the conversation. This example assumes that the home screen is not enabled, which means that the web chat initiates each new conversation with an empty message to trigger a greeting from the assistant.

   For this example, we are using the hardcoded user name `Cade`. In a production assistant, you might retrieve the customer's name from a user profile on your website.

   ```
   $ function preSendHandler(event) {
     // Only do this on messages that request the welcome message.
     if (event.data.input && event.data.input.text === '') {
       event.data.context.skills['actions skill'] = event.data.context.skills['actions skill'] || {};
       event.data.context.skills['actions skill'].skill_variables = event.data.context.skills['actions skill'].skill_variables || {};
       event.data.context.skills['actions skill'].skill_variables.User_Name = 'Cade';
     }
   }
   ```

2. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `pre:send` event, registering the `preSendHandler()` function as the callback.

   ```
   $ instance.on({ type: 'pre:send', handler: preSendHandler });
   ```

The assistant actions can now access the `User_Name` variable throughout the conversation.

For complete working code, see the Setting context variables for watsonx Assistant web chat example.

Tutorial: Interacting with the host web page

You can use custom responses and events to enable the web chat to interact with the web page where it appears.

For example, customers might use your assistant to find information they need to complete a form. Rather than expecting the customer to then copy this information manually to the form, you can have the web chat automatically complete the information.

Note: For a complete, working version of the example in this tutorial, see Page interactions for watsonx Assistant web chat.

This example uses a custom response to render a button in the web chat that populates a form field with the customer's account number:

1. Create a handler for the `customResponse` event. This handler renders a custom button and creates a click handler for it. The click handler uses the `Document.querySelector()` method to interact with the DOM and complete a form field with the customer's account number.

   Note: This example uses the hardcoded account number `1234567`. In a typical production assistant, your assistant would retrieve this value from a session variable or query it from an external system.

   ```
   $ function customResponseHandler(event) {
     const { element } = event.data;

     const button = document.createElement('button');
     button.type = 'button';
     button.innerHTML = 'Enter account number';
     button.addEventListener('click', () => {
       // Look for the account number element in the document, and enter the account number.
       document.querySelector('#account-number').value = '1234567';
     });

     element.appendChild(button);
   ```

```
}
```

2. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `customResponse` event, registering the handler as the callback. This enables the assistant to send a custom response that displays the button for filling in the account number.

```
$ instance.on({
  type: 'customResponse',
  handler: (event, instance) => {
    const { message } = event.data;
    if (message.user_defined &&
        message.user_defined.user_defined_type === 'fill_account_number') {
          accountNumberResponseHandler(event, instance);
        }
  },
});
```

Note: In this example, we are checking the custom `user_defined_type` property of the custom response and calling the `accountNumberResponseHandler()` function only if the specified type is `fill_account_number`. This is an optional check that shows how you might use a custom property to define multiple different custom responses (each with a different value for `user_defined_type`).

For complete working code, see the Page interactions for watsonx Assistant web chat example.

Tutorial: Authenticating a user in the middle of a session

If you have web chat security enabled, you must set a user ID for the customer at the beginning of the session as part of the JSON Web Token (JWT) you use to sign messages. For users who are not authenticated, this is typically a generated ID, and it cannot be changed after the JWT is created. However, you can use a private variable to authenticate a user later in the session.

With web chat security enabled, the user ID associated with each message is based on the `sub` claim in the JWT payload. This value must be set at the beginning of the session, when the JWT is created, and cannot be changed during the life of the session. For unauthenticated (anonymous) users, you would typically use a generated ID, saved to a cookie, to ensure that each unique customer is counted only once for billing purposes.

However, you might want your customers to be able to authenticate in the middle of a session (for example, to complete an action that updates the user's account information). Because the generated user ID in the `sub` claim cannot be changed, you need another way to authenticate the user securely. You can do this by storing the customer's actual authenticated user ID as a private variable in the user payload of the JWT. (You could store the user ID in an ordinary context variable, but this would not be secure, because such variables can be modified.)

Note: For a complete, working version of the example described in this tutorial, see Enabling security for watsonx Assistant web chat (complex).

Note: For a web chat version with minimal codes to enable security, which does not have encrypted payloads and user changes during the session, see Enabling security for watsonx Assistant web chat (simple).

This example in this tutorial, which is based on an Express server for Node.js, shows how to start a session with an anonymous user ID and then authenticate the user during the session.

1. Create a function called `getOrSetAnonymousID()` that generates a unique anonymous user ID for each customer and stores it in a cookie (or, if the cookie already exists, uses the stored user ID).

   Tip: Use a cookie that lasts for at least 45 days. If you do not store the user ID for more than 30 days, the same customer might be counted as multiple different users during the same billing period. (This can still happen if the same user deletes the cookie or uses a different browser.)

```
$ function getOrSetAnonymousID(request, response) {
  let anonymousID = request.cookies['ANONYMOUS-USER-ID'];
  if (!anonymousID) {
    anonymousID = `anon-${uuid()}`;
  }

  response.cookie('ANONYMOUS-USER-ID', anonymousID, {
    expires: new Date(Date.now() + 1000 * 60 * 60 * 24 * 45), // 45 days.
    httpOnly: true,
  });
```

```
  return anonymousID;
}
```

1. In the function you use to create a JWT, use the anonymous ID returned from the `getOrSetAnonymousID()` function as the value of the `sub` claim. This sets the value of the user ID that will be used to uniquely identify the customer for billing purposes.

   In addition, retrieve any value from the `SESSION_INFO` cookie, which we will use to store authenticated login information. If a value exists, store it in the `user_payload` private claim of the JWT. (If the user has not yet authenticated, this cookie does not yet exist.)

```
$ const jwtContent = {
  sub: anonymousUserID,
  user_payload: {
    name: 'Anonymous',
    custom_user_id: anonymousUserID,
  },
};

if (sessionInfo) {
  jwtContent.user_payload.name = sessionInfo.userName;
  jwtContent.user_payload.custom_user_id = sessionInfo.customUserID;
}
```

1. Create a function to handle user authentication. In our example, we are using a simple `authenticate()` function that sets a hardcoded user ID, but in a real application the user ID would probably be retrieved from a database after a secure authentication check. Store the user information in the `SESSION_INFO` cookie.

```
$ function authenticate(request, response) {

  const userInfo = {
    userName: 'Cade',
    customUserID: 'cade-id',
  };

  response.cookie('SESSION_INFO', JSON.stringify(userInfo), { encode: String });

  response.send('Ok');
}
```

1. When a user logs in, call the `authenticate()` function to store the user information in the `SESSION_INFO` cookie. Then call the `createJWT()` function to regenerate the JWT, using the updated session info to populate the `user_payload` claim.

   In [our example](), authentication is simulated with a simple button click. The same button also simulates logging out by deleting the cookie:

```
$ async function onClick() {
  if (getCookieValue('SESSION_INFO')) {
    document.cookie = 'SESSION_INFO=; Max-Age=0';
  } else {
    await fetch('http://localhost:3001/authenticate');
  }

  const result = await fetch('http://localhost:3001/createJWT');
  const newToken = await result.text();

  webChatInstance.updateIdentityToken(newToken);

  updateUI();
}
```

   The anonymous ID in the `sub` claim will continue to be used to track the customer for billing purposes, but now the customer's real user ID is stored separately in the user payload.

2. In your actions, you can now access the customer's real user ID by referencing the `user_payload` private context variable:

```
$ ${system_integrations.chat.private.user_payload}.custom_user_id
```

For complete working code, see the [Enabling security for watsonx Assistant web chat]() example.

⚠ **Important:** If you are required to comply with GDPR requirements, you might need to persistently store any generated anonymous user IDs, especially for anonymous users who later log in with user credentials. Storing these user IDs makes it possible for you to later delete all data associated with an individual customer if requested to do so.

Tutorial: Customizing the size and position of the web chat

This tutorial shows how you can change the size and position of the web chat by rendering it in a custom element.

Note: For a complete, working version of the example described in this tutorial, see Custom elements for watsonx Assistant web chat.

By default, the web chat interface on your website is rendered in a host `div` element that is styled to appear in a fixed location on the page. If you want to change the size or position of the web chat, you can specify a custom element as the host location for the web chat. This host element is used as the location for both the main web chat interface and for the web chat launcher (unless you are using a custom launcher).

When you use a custom element, you also take control of showing and hiding the web chat when it is opened or closed (such as when the customer clicks the launcher icon or the minimize button). This gives you the opportunity to apply additional effects, such as opening and closing animations. You can control showing and hiding the main window by using the addClassName() and removeClassName() functions.

To use a custom element, follow these steps:

1. In your website code, define the custom element where you want the web chat to be rendered. There are many ways of doing this, depending on the framework you are using. A simple example is to define an empty HTML element with an ID:

```
$ <div id="WebChatContainer"></div>
```

2. Get a reference to your custom element so you can reference it in the web chat configuration. To get a reference, use whatever mechanism makes sense for the library you are using. For example, you can save the reference returned from `document.createElement()`, or you can use a query function to look up the element in the DOM. This example looks up the element using the ID we assigned to it:

```
$ const customElement = document.querySelector('#WebChatContainer');
```

3. In the web chat embed script, set the `element` property, specifying the reference to your custom element.

```
$ window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",

  // The important piece.
  element: customElement,

  onLoad: function(instance) {
    instance.render();
  }
};
```

4. Make sure that the main web chat window is hidden by default. You can do this in the `onLoad` event handler, after `render` has been called. You must also add handlers to listen for the `view:change` event so the customer can open and close the web chat after the page loads. In our example, we are using a CSS class called `HideWebChat` to do this (see the full example for the definition of this class):

```
$ function onLoad(instance) {
  instance.render();
  instance.on({ type: 'view:change', handler: viewChangeHandler });
  instance.elements.getMainWindow().addClassName('HideWebChat');
}
```

5. Create handlers to hide and show the main web chat window in response to the `view:change` event. This example simply shows and hides the element; the full example shows how you can add animation effects.

```
$ function viewChangeHandler(event, instance) {
  if (event.newViewState.mainWindow) {
    instance.elements.getMainWindow().removeClassName('HideWebChat');
  } else {
    instance.elements.getMainWindow().addClassName('HideWebChat');
```

```
 }
}
```

For complete working code, see the Custom elements for watsonx Assistant web chat example.

Tutorial: Displaying a pre-chat or post-chat form

This tutorial shows how you can display pre-chat form before the web chat opens, or a post-chat form that opens after the web chat closes.

Note: For a complete, working version of the example described in this tutorial, see Pre and post-chat forms for watsonx Assistant web chat.

If you want to gather information from your customers before starting a chat session, you can display a pre-chat form before opening the web chat. Similarly, you might want to display a form after the web chat closes (for example, a customer satisfaction survey). You can use the same approach for either situation.

When the web chat is opened or closed, it fires an event that you can subscribe to. In your event handler, you can use the custom panels feature to open a panel with custom content.

By returning a promise that is resolved when the custom panel closes, you can pause the process of opening or closing the web chat until after the customer completes the form.

Tip: This example shows how to create a pre-chat form. To create a post-chat form, follow the same steps, but use `!event.newViewState.mainWindow`.

To display a pre-chat form, follow these steps:

1. Create a handler for the `view:change` event, which is fired when one of the view in web chat changes (such as when the main window opens). This handler uses the `customPanels.getPanel()` instance method to open a custom panel that contains the pre-chat form.

   Tip: Your handler should return a promise that resolves when the custom panel is closed. This prevents the web chat main window from opening until after the pre-chat form is completed.

   ```
   function viewChangeHandler(event, instance) {
    const mainWindowOpening = !event.oldViewState.mainWindow && event.newViewState.mainWindow;
    if (mainWindowOpening) {
     return new Promise((resolve) => {
       promiseResolve = resolve;

       createOpenPanel(instance);
       const customPanel = instance.customPanels.getPanel();
       customPanel.open({ hidePanelHeader: true,
                          disableAnimation: true });
     });
    }
   ```

2. In your `onLoad` event handler, use the `on()` instance method to subscribe to the `view:change` event, registering the handler as the callback.

   ```
   instance.on({ type: 'view:change', handler: viewChangeHandler });
   ```

3. Create a function that creates the pre-chat form you want to show inside the custom panel. Make sure you resolve the promise when the user closes the panel.

   ```
   function createOpenPanel(instance) {
    const customPanel = instance.customPanels.getPanel();
    const { hostElement } = customPanel;

    hostElement.innerHTML = `
      <div class="PreChatForm">
        ... // Content of pre-chat form
      </div>
    `;

    const okButton = hostElement.querySelector('#PreChatForm__OkButton')
    okButton.addEventListener('click', () => {
   ```

```
    customPanel.close();
    promiseResolve();
  });
}
```

Tip: A React version of this example is also available.

For complete working code, see Pre and post-chat forms for watsonx Assistant web chat example.

Tutorial: Customizing text elements based on the current page

This tutorial shows how you can dynamically customize the launcher or home screen text based on the page the user is currently viewing.

Note: For a complete, working version of the example described in this tutorial, see Change launcher and home screen text for watsonx Assistant web chat.

The launcher text and home screen greet the user and suggest what the user might want to do. Rather than showing the same text all the time, you might want to customize this text based on the page on your website the user is currently viewing. For example, instead of a generic greeting, a customer viewing a page about credit cards might see text that is specific to credit card services.

Tip: Although this example shows how to adapt the text based on the current page, you can use the same basic approach to adapt the text based on any client condition (such as the time of day or the user's geographical location).

To change the launcher or home screen text elements based on the current page the user is viewing, follow these steps:

1. Determine what page the user is currently viewing. Depending on the design of your application, there are various ways to do this, but one simple mechanism is to check the value of a URL query parameter (in this example, `page`):

   ```
   $ const page = new URLSearchParams(window.location.search).get('page');
   ```

2. When the web chat is loaded, check this value and conditionally set the text based on which page is being viewed. You can use any condition that makes sense for your application. In this example, we are simply checking the value of the `page` query parameter:

   ```
   if (page === 'cards') {
     // Update the launcher and home screen with text that is
     // specific to credit cards.
     ...
   } else if (page === 'account') {
     // Update the launcher and home screen with text that is
     // specific to the user's account.
     ...
   }
   ```

3. Change the launcher text using the `updateLauncherGreetingMessage()` instance method. This changes the text that is displayed on the launcher (by default after 15 seconds).

   ```
   $ instance.updateLauncherGreetingMessage("I see you're interested in credit     cards! Let me know if I can help.");
   ```

4. Change the home screen configuration using the `updateHomeScreenConfig()` instance method:

   - Make sure the home screen is enabled.
   - Change the greeting based on the page the user is viewing.
   - Configure the buttons shown by the home screen so they offer options that are appropriate for the page the user is viewing.

   ```
   $ instance.updateHomeScreenConfig({
     is_on: true,
     greeting: 'What can I tell you about credit cards?',
     starters: {
       is_on: true,
       buttons: [
         { label: 'Card interest rates' },
         { label: 'Cards with rewards' },
   ```

```
    { label: 'Business cards' }
   ]
  }
});
```

For complete working code, see the Change launcher and home screen text for watsonx Assistant web chat example.

Tutorial: Customizing the greeting action based on the current page

This tutorial shows how you can dynamically customize the greeting action that is triggered by the web chat based on the page the user is currently viewing.

Note: For a complete, working version of the example described in this tutorial, see Select greeting action for watsonx Assistant web chat.

If you do not have the home screen enabled, the default behavior when the web chat opens is to send an empty message to the assistant to start the conversation. This empty message triggers the *Greet customer* action, which typically sends a welcome message.

Rather than starting the conversation with a generic greeting, you might want your users to see a prompt that is specific to the page they are already viewing. For example, if a user has already navigated to a page about credit cards and then opens the web chat, you might want to start the conversation with a message that is specific to credit cards.

Tip: Although this example shows how to adapt the text based on the current page, you can use the same basic approach to adapt the text based on any client condition (such as the time of day or the user's geographical location).

To change the greeting action based on the current page the user is viewing, follow these steps:

1. Determine what page the user is currently viewing. Depending on the design of your application, there are various ways to do this, but one simple mechanism is to check the value of a URL query parameter (in this example, `page`):

   ```
   $ const page = new URLSearchParams(window.location.search).get('page');
   ```

2. Create a handler for the `pre:send` event, which is fired before a message is sent to the assistant. This handler starts by checking the `is_welcome_request` property to determine whether the message being sent is the empty message that starts the conversation by triggering the *Greet customer* action.

   If it is, the handler then checks the currently displayed page, and it replaces the outgoing empty message with a message that will trigger an action that is specific to the page. (This example only shows setting the message to `Credit Cards`, but additional `if` conditions could customize the message for other pages.)

   ```
   $ function preSendHandler(event) {
    if (event.data.history && event.data.history.is_welcome_request) {
     if (page === 'cards') {
      event.data.input.text = 'Credit Cards';
     }
    }
   }
   ```

3. In your `onLoad` event handler, use the `once()` instance method to subscribe to the `pre:send` event, registering the `preSendHandler()` function as the callback. (We can use `once()` instead of `on()` because this handler needs to be called only for the first message in the session.)

   ```
   $ instance.once({ type: 'pre:send', handler: preSendHandler});
   ```

For complete working code, see the Select greeting action for watsonx Assistant web chat example.

Deploying for phone

Integrating with phone

IBM Cloud Plus

Deprecated: IntelePeer free phone number service for watsonx Assistant
Effective 31 July 2025, the IntelePeer free phone number service is discontinued for watsonx Assistant. After this date, phone numbers that use this service no longer connect, and users can't make or receive calls. To ensure continued service, migrate your phone numbers to Twilio or another telephony provider. Update your phone numbers and adjust your integration settings in watsonx Assistant. For assistance with the transition, contact support at https://www.ibm.com/mysupport/.

Note: You can enable the streaming feature in conversational search for the phone integrations in your assistant. This helps your assistant to stream the responses while generating them. For more information see, Conversational search.

Adding the phone integration to your assistant makes your assistant available to customers over the phone.

If an end user asks to speak to a person, the phone integration can transfer the call to an agent. Supported live agent and contact center integrations:

- Genesys
- Twilio Flex
- NICE CXone
- Bring your own

There are several ways to add the phone integration to your assistant:

- You can generate a free phone number that is automatically provisioned from IntelePeer. This is available only with new phone integrations. If you have an existing phone integration, you must delete it and create a new one to switch to a free phone number.
- You can connect to a contact center with live agents. For more information about setting up the integration, see Integrating with phone and NICE CXone contact center.
- You can use and connect an existing number by configuring a Session Initiation Protocol (SIP) trunk from a provider such as Genesys, IntelePeer, or Twilio.

A SIP trunk is equivalent to an analog telephone line, except it uses Voice over Internet Protocol (VoIP) to transmit voice data and can support multiple concurrent calls. The trunk can connect to the public switched telephone network (PSTN) or your company's on-premises private branch exchange (PBX).

When a customer makes a phone call using the telephone number connected to your assistant, the phone integration makes it possible for your assistant to answer. The integration converts output from your assistant into voice audio by using the IBM Watson® Text to Speech service, and the audio is sent to the telephone network through the SIP trunk. When the customer replies, the voice input is converted into text by using the IBM Watson® Speech to Text service.

This feature is available only to Plus or Enterprise plan users.

Tip: Depending on the architecture of your existing telephony infrastructure, there are multiple ways you might integrate it with watsonx Assistant. For more information about common integration patterns, read the blog post Hey Watson, can I have your number? on Medium.

Set up the integration

You must have Manager role access to the instance and Viewer role access to the resource group to complete setup. For more information about access levels, see Managing access.

To set up the integration:

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you will see a tile for **Phone**.
2. On the **Phone** tile, click **Add**.
3. On the pop-up window, click **Add** again.
4. Choose whether to generate a free phone number for your assistant, integrate with your contact center, or connect to an existing SIP trunk:
    - To generate a free phone number for your assistant, click **Generate a free phone number**.

> **Note:** Generating a free phone number is supported only for watsonx Assistant instances in the Dallas and Washington DC data centers.

- To integrate with a contact center, click **Integrate with your contact center** .
- To use an existing phone number you have already configured with a SIP trunk provider, click **Use an existing phone number with an external provider**.

Click **Next**.

5. If you are integrating with a contact center, follow the instructions to configure the contact center. Skip this step if you are generating a free phone number.

   a. On the **Select contact center** page, select the tile of the connect center you would like to use.

   b. On the **Connect to contact center** page, enter the required information. There is a **Test Connection** button on the page to validate the connection. Click **Next**.

6. If you are using an existing phone number, follow the instructions to configure the SIP trunk. Skip this step if you are generating a free phone number.

   a. On the **Bring your own SIP trunk** page, copy the SIP URI and assign it to your SIP trunk. Click **Next**.

7. On the **Phone number** page (only for **Integrate with your contact center** and **Use an existing phone number with an external provider**), specify the phone number of the SIP trunk. Specify the number by using the international phone number format: `+1 958 555 0123` . Do not surround the area code with parentheses.

   Currently, only one primary phone number can be added during initial setup of the phone integration. You can add more phone numbers in the phone integration settings later.

   Click **Next**.

8. On the **Speech to Text** page, select the instance of the Speech to Text service you want to use for the phone integration.

   - If you have existing Speech to Text instances, select the instance you want to use from the list.
   - If you do not have any existing Speech to Text instances, click **Create new instance** to create a new Plus instance.

9. In the **Choose your Speech to Text language model** field, select the language model you want to use.

   The list of language models is automatically filtered to use the same language as your assistant. To see all language models, toggle the **Filter models based on assistant language** switch to **Off**.

   > **Note:** If you created specialized custom models that you want your assistant to use, choose the Speech to Text service instance that hosts the custom models now, and you can configure your assistant to use them later. The Speech to Text service instance must be hosted in the same location as your watsonx Assistant service instance. For more information, see Using a custom language model .

   For more information about language models, see Languages and models in the Speech to Text documentation.

   Click **Next**.

10. On the **Text to Speech** page, select the instance of the Text to Speech service you want to use for the phone integration.

    - If you have existing Text to Speech instances, select the instance you want to use from the list.
    - If you do not have any existing Text to Speech instances, click **Create new instance** to create a new Standard instance.

11. In the **Choose your Text to Speech voice** field, select the voice you want to use.

    The list of voices is automatically filtered to use the same language as your assistant. To see all voices, toggle the **Filter voices based on assistant language** switch to **Off**.

    For more information about voice options, and to listen to audio samples, see Languages and voices in the Text to Speech documentation.

    Click **Next**.

> **Important:** Any speech service charges incurred by the phone integration are billed with the watsonx Assistant service plan as *voice add-on* charges. After the instances are created, you can access them directly from the IBM Cloud dashboard. Any use of speech instances that occurs outside of your assistant is charged separately as speech service usage costs.

The phone integration setup is now complete. On the **Phone** page, you can click the tabs to view or edit the phone integration.

> **Note:** If you chose to generate a free telephone number, your new number is displayed on the **Phone number** tab immediately. However, provisioning the new number might take several minutes.

Adding more phone numbers

If you are using existing phone numbers you configured via a SIP trunk provider, you can add multiple numbers to the same phone integration.

> **Note:** If you generated a free phone number, you cannot add more numbers.

To add more phone numbers:

1. In the phone integration settings, go to the **Phone number** tab.
2. Use one of the following methods:
    - To add phone numbers one by one, click **Add phone number** in the table, and enter the phone number along with an optional description. Click the **Add** button to save the number.
    - To import a set of phone numbers that are stored in a comma-separated values (CSV) file, click the *Upload a CSV file* icon (![Add phone number][images/phone-integ-import-number.png]), and then find the CSV file that contains the list of phone numbers.

      The phone numbers you upload will replace any existing numbers in the table.

Setting up live agent escalation

If you want your assistant to be able to transfer a conversation to a live agent, you can connect your phone integration to a contact center. For more information, see instructions for the supported platform:

- Genesys
- Twilio Flex

Phone integration limits

Any speech service charges incurred by the phone integration are included as *Voice add-on* charges in your watsonx Assistant service plan usage. The Voice add-on use is charged separately and in addition to your service plan charges.

Plan usage is measured based on the number of monthly active users, where a user is identified by the caller's unique phone number. An MD5 hash is applied to the phone number and the 128-bit hash value is used for billing purposes.

The number of concurrent calls that your assistant can participate in at one time depends on your plan type.

Plan	Concurrent calls
Enterprise	1,000
Plus	100
Trial	5

Plan details

Phone integration configuration

IBM Cloud

After you have set up the phone integration for your assistant, you can modify the phone integration settings to customize the call behavior.

Handling call and transfer failures

You can configure the phone integration to transfer the caller to a live agent if the phone connection fails for any reason. To transfer the caller to a human automatically, go to the **Advanced** tab in the phone integration settings, and make the following configuration selections:

- **SIP target when a call fails** : Add the SIP endpoint for your support agent service. Specify a SIP or telephone URI for a general call queue that can redirect requests to other queues. For more information, see Configuring a backup service desk solution .
- **Call failure message**: Add the message you want the assistant to say to a caller before it transfers the call to a live agent.

If, after you transfer the call to a human, the connection to a live agent fails for any reason, you can configure what to do.

- **Transfer failure message**: Add the message you want the assistant to say to a caller if transfer to a live agent fails. The message can be up to 150 characters in length.
- **Disconnect call on transfer failure** : Choose whether to disconnect the call after the failure message. This option is enabled by default. If this option is disabled, when a call transfer fails, your assistant can disconnect or process a different action.

  If you choose to leave a call connected despite a transfer failure, watsonx Assistant initiates a new turn to determine the next step. It's important that the Assistant be configured with an Action or webhook that can handle this scenario.

Note: The phone integration supports disaster recovery by providing the ability to do a fast failover to another region instead of routing the call to a live agent when a service outage occurs. This is accomplished by sending a SIP 503 response to the upstream SIP trunking provider, instead of auto referring the call to a live agent when failures happen during the setup of a call. This 503 response can then be used by the SIP trunking provider to reroute the call to another region. If you want to take advantage of this capability, open a service ticket against the watsonx Assistant service instance that requires disaster recovery.

Secure the phone connection

You can add security to the phone connection by going to the **Advanced options** tab in the phone integration settings, and selecting one or both of the following options:

- **Force secure trunking**: Select this option to use Secure Real-Time Transfer Protocol (SRTP) to secure the audio that is transmitted over the phone. For more information about RTP, see Call routing details.
- **Enable SIP authentication**: Select this option if you want to require SIP digest authentication.

  When SIP authentication is required, all inbound traffic (meaning requests from the SIP provider to your assistant) is authenticated using SIP digest authentication, and must be sent using Transport Layer Security (TLS). If this option is selected, the SIP digest user name and password must be configured, and the SIP trunk being used to connect to Assistant must be configured to use only TLS.

  Important: If you use Twilio as your SIP trunk provider, you cannot enable SIP authentication for outbound SIP trunks to watsonx Assistant.

Applying advanced SIP trunk configuration settings

To configure how your assistant interacts with a SIP trunk from an external provider, go to the **SIP trunk** tab in the phone integration settings, and update the following options in the **SIP trunking integration** section:

- **SIP INVITE headers to extract** : List the headers that you want your assistant to use.

  The SIP INVITE request can include metadata about the call in headers that can be extracted and sent to your assistant using context variables. For example, many companies use Interactive Voice Response (IVR) systems that pass information about an incoming call by using SIP headers. If you want to make use of any of these headers, list the header names here.

  The specified headers, if present in the request, are stored in the context variable `sip_custom_invite_headers`, along with other related metadata that is automatically extracted from the SIP INVITE. This variable is an array in which each key/value pair represents a header from the request, as in this example:

```
$ {
  "input": {
    "text": "",
      ...
  },
  "context" : {
    "global" : {...},
    "skills" : {...},
    "integrations" : {
      "voice_telephony": {
        "private":{
          "user_phone_number":"+18594213456",
        },
        "sip_call_id": "Aob2-2743-5678-1234",
```

```
      "assistant_phone_number":"+18882346789",
      "sip_custom_invite_headers": {
        "X-customer-name": "my_name",
        "X-account-number": "12345"
      }
    }
  }
 }
}
```

You can then reference these headers in your assistant. For example, you might check the header value in a step condition to determine the next step. You can also use these headers when searching the assistant logs; for example, you might search for a custom header to find all the messages associated with particular account.

- **Disable the ring that callers hear while the assistant is contacted** : Choose whether you want the caller to hear a signal that indicates the assistant is being contacted.

  A `180 Ringing` response is sent from the assistant back to the SIP trunk provider while your assistant processes the incoming call invitation. The ringing response is sent by default.

- **Don't place callers on hold while transferring to a live agent** : Choose whether the phone integration puts the caller on hold.

  If your SIP trunk provider manages holds, disable this feature. For example, some SIP trunk providers prefer to have the assistant send a SIP REFER request, so they can put the call on hold themselves.

For more information about the SIP protocol, see RFC 3261 and about the RTP protocol, see RFC 3550.

Configuring a backup service desk solution

When you use the phone integration as the first line of assistance for customers, it's a good idea to have a live agent backup available. You can design your assistant to transfer a call to a human in case the phone connection fails, or if a user asks to speak to someone.

Your company might already have one or more phone numbers that connect to an automatic call dispatcher (ACD), which can queue callers until an appropriate agent is available. If not, choose a service desk solution to use as your backup.

A conversation cannot be transferred from one integration type to another. For example, if you use the web chat integration with service desk support, you cannot transfer a phone call to the service desk that is set up for the web chat.

You must provide the service desk SIP URI for the service desk support solution you use. You must specify this information in your assistant when you enable a call transfer from a dialog node or action step. For more information, see Transferring a call to a live agent.

Optimize your actions for phone interaction

For the best customer experience, design your dialog with the capabilities of the phone integration in mind:

- Do not include HTML elements in your action responses. To add formatting, use Markdown. For more information, see Formatting responses.
- You can use a search extension to include search results in actions that the phone integration will read. When search results are returned, the phone integration reads the introductory message (for example, `I found this information that might be helpful`), and then the body of only the first search result.

  The entire search response (meaning the introductory message plus the body of the first search result) must be less than 5,000 characters long or the response will not be read at all. Be sure to test the search results that are returned and curate the data collection that you use as necessary.

For more information about using the search integration, see Leveraging existing help content.

For more information about how to implement common actions from your dialog, see Handling phone interactions.

Creating a SIP trunk

If you do not use the option to generate a free phone number, you must set up the SIP trunk that is used by the phone integration. Find a provider and create a SIP trunk account, for which you must pay per usage.

You can set up a SIP trunk in the following ways:

- Setting up a Twilio SIP trunk
- Use other third-party providers
- Bring your own SIP trunk
- Migrate from Voice Agent with Watson

Setting up a Twilio SIP trunk

Before you begin the setup of a Twilio SIP Trunk, do the following prerequisite steps:

- Create a Twilio account on the Twilio website.
- Create a SIP trunk

If you already created a SIP trunk, follow steps in Configure a SIP Trunk.

Create a SIP trunk

1. Log in to your Twilio console and go to the **Elastic SIP Trunking** section.

   Note: If you do not see *Elastic SIP Trunking*, do the following:

   a. Search for `Elastic SIP Trunking` in the **Search Bar**.
   b. Click **Elastic SIP Trunking Dashboard** .

2. In the left navigation menu, go to **Overview** and click **Get Started**.
3. Click **Create New Trunk** in the *Trunks* navigation bar.
4. Enter a name for your SIP trunk in the **FRIENDLY NAME** field.
5. Click **Create** button. A **Trunk SID** is assigned once you create a new trunk.

For configuration of a SIP trunk, follow steps in Configure a SIP Trunk.

Configure the SIP trunk

To configure a SIP trunk do the following:

1. From the *Elastic SIP Trunking Dashboard* page, go to **Elastic SIP Trunking**.
2. Click **Manage** in the left navigation menu.
3. Click **Trunks** and select the SIP trunk that you created.
4. Click **Origination** to configure its settings.
5. To add the origination SIP URI, click the **Add new Origination URI** button and provide values for the following fields:
   - **Origination SIP URI** - You can get the SIP URI for your phone integration on the watsonx Assistant's phone integration configuration page. To do this in watsonx Assistant, launch the tooling and *Create Assistant*. Choose **Add integration** and select **Phone**. From your assistant, copy the **SIP URI** and paste it in **Origination SIP URI** field of Twilio.
   - **Priority** - Priority ranks the importance of the URI. A lower number represents the highest importance.
   - **Weight** - Weight is used to determine the share of load when more than one URI has the same priority. The higher the value, the more load a URI is given.
   - **Enabled** - You need to switch the *Enabled* toggle to `Yes`. This means that origination SIP URI is enabled.
6. If you plan to support call transfers, enable Call Transfer (SIP REFER) in your SIP trunk. If you expect to transfer calls to the public switched telephone network (PSTN), also enable PSTN Transfer on your trunk.
7. Select **Numbers** from the navigation bar for your SIP trunk, and then do one of the following things:

   a. Click **Add a number** and then **Buy a Number***.
   b. If you already have a number, you can click *Add a number* and then *Add an Existing Number*.

If you use a Lite or Trial Twilio account for testing purposes, then be sure to verify the transfer target. For more information, see the Twilio documentation.

You cannot enable SIP authentication if you choose Twilio as your SIP trunk provider. Twilio doesn't support SIPS for originating calls.

Using other third-party providers

You can ask for help setting up an account with another SIP trunk provider by opening a support request.

IBM has established relationships with the following SIP trunk providers:

- Five9
- Genesys

- Vonage
- Voximplant

The SIP trunk provider sets up a SIP trunk for your voice traffic, and manages access from allowed IP addresses. Most of the major SIP trunk providers have existing relationships with IBM. Therefore, the network configuration that is required to support the SIP trunk connection typically can be handled for you with minimal effort.

1. Create an IBM Cloud case.
2. In the **Topic** field, enter `watsonx Assistant`.
3. In the **Subtopic** drop-down list, select `Phone & SMS Integration`.
4. In the **Subject** field, enter `SIP trunk provider setup for watsonx Assistant`.
5. Include the following information in the description:
    - Company Name
    - Your IBM Cloud account ID
    - Your watsonx Assistant service name
    - Network diagram with IP address or SIP trunk provider information

Bring your own SIP trunk

If you choose to use a SIP trunk carrier that IBM does not have an established relationship with, you can do so.

The following table lists the fully qualified domain names and IP addresses that are used for SIP connections.

Location	Domain names	IP addresses
Dallas	public.0001.voip.us-south.assistant.watson.cloud.ibm.com public.0002.voip.us-south.assistant.watson.cloud.ibm.com public.0003.voip.us-south.assistant.watson.cloud.ibm.com	67.228.108.82 169.63.5.162 150.239.30.146
Frankfurt	public.0001.voip.eu-de.assistant.watson.cloud.ibm.com public.0002.voip.eu-de.assistant.watson.cloud.ibm.com public.0003.voip.eu-de.assistant.watson.cloud.ibm.com	161.156.178.162 169.50.56.146 149.81.86.82
London	public.0001.voip.eu-gb.assistant.watson.cloud.ibm.com public.0002.voip.eu-gb.assistant.watson.cloud.ibm.com public.0003.voip.eu-gb.assistant.watson.cloud.ibm.com	158.176.120.162 141.125.102.34 158.175.99.34
Seoul	public.0001.voip.kr-seo.assistant.watson.cloud.ibm.com	
Sydney	public.0001.voip.au-syd.assistant.watson.cloud.ibm.com public.0002.voip.au-syd.assistant.watson.cloud.ibm.com public.0003.voip.au-syd.assistant.watson.cloud.ibm.com	168.1.47.2 135.90.86.50 168.1.106.130
Tokyo	public.0001.voip.jp-tok.assistant.watson.cloud.ibm.com public.0002.voip.jp-tok.assistant.watson.cloud.ibm.com public.0003.voip.jp-tok.assistant.watson.cloud.ibm.com	165.192.69.82 128.168.105.178 161.202.149.162
Washington, DC	public.0001.voip.us-east.assistant.watson.cloud.ibm.com public.0002.voip.us-east.assistant.watson.cloud.ibm.com public.0003.voip.us-east.assistant.watson.cloud.ibm.com	52.116.100.158 169.61.70.162 169.59.136.194

SIP network information

Migrating from Voice Agent with Watson

If you created an IBM® Voice Agent with Watson service instance in IBM Cloud to enable customers to connect to an assistant over the phone, consider using the phone integration instead. You can use the same SIP account and phone number that you configured for use with Voice Agent with Watson in the phone integration.

The phone integration provides a more seamless integration with your assistant. However, the integration currently does not support the following functions:

- Outbound calling
- Configuring backup locations
- Event forwarding to save call detail reports in the IBM Cloudant for IBM Cloud database service
- Reviewing the usage summary page. Use IBM Log Analysis instead. For more information, see Viewing logs.

To migrate from Voice Agent with Watson to the watsonx Assistant phone integration, complete the following steps:

1. From the Voice Agent with Watson page, copy the phone number or numbers that you used for your SIP account.
2. When you set up the watsonx Assistant phone integration, add the phone number or set of numbers that you copied in the previous step.
3. From the phone integration setup page, copy the SIP uniform resource identifier (URI).
4. In your SIP trunk account, replace the Voice Agent with Watson URI that you specified previously with the URI that you copied from the phone integration setup page in the previous step.

   For example, if you use a Twilio SIP trunk, you would add the assistant's SIP uniform resource identifier (URI) to the Twilio **Origination SIP URI** field.
5. If your SIP trunk provider is not already allowlisted with the watsonx Assistant region you are migrating to, follow these instructions to get access to your SIP trunk.

Call routing details

Incoming calls to your assistant follow this path:

1. A customer calls the customer support phone number that is managed by your Session Initiation Protocol (SIP) trunk provider.
2. The SIP trunk service sends a SIP `INVITE` request to your assistant's phone integration to establish a connection.
3. The phone integration connects to the speech services that are required to support the interaction.
4. After the services are ready, the connection is established, and audio is sent over the Real-time Transport Protocol (RTP).

   RTP is a network protocol for delivering audio and video over IP networks.
5. The greeting action of the assistant is processed. The response text is sent to the Text to Speech service to be converted to audio and the audio is sent to the caller.
6. When the customer says something, the audio is converted to text by the Speech to Text service and is sent to your assistant for evaluation.
7. The assistant processes the input and calculates the best response. The response text from the assistant is sent to the Text to Speech service to be converted to audio and the audio is sent back to the caller over the existing connection.
8. If the caller asks to speak to a person, the assistant can transfer the person to a call center. A SIP `REFER` request is sent to the SIP trunk provider so it can transfer the call to the call center SIP URI that is specified in the dialog node where the transfer action is configured.
9. When one of the participants of the call hangs up, a SIP `BYE` request is sent to the other participant.

Integrating with phone and Genesys Cloud

IBM Cloud Plus

You can use the phone integration to help your customers over the phone and transfer them to live agents inside of Genesys Cloud. If in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Genesys Cloud agent.

Before you begin

To use this integration pattern, you need:

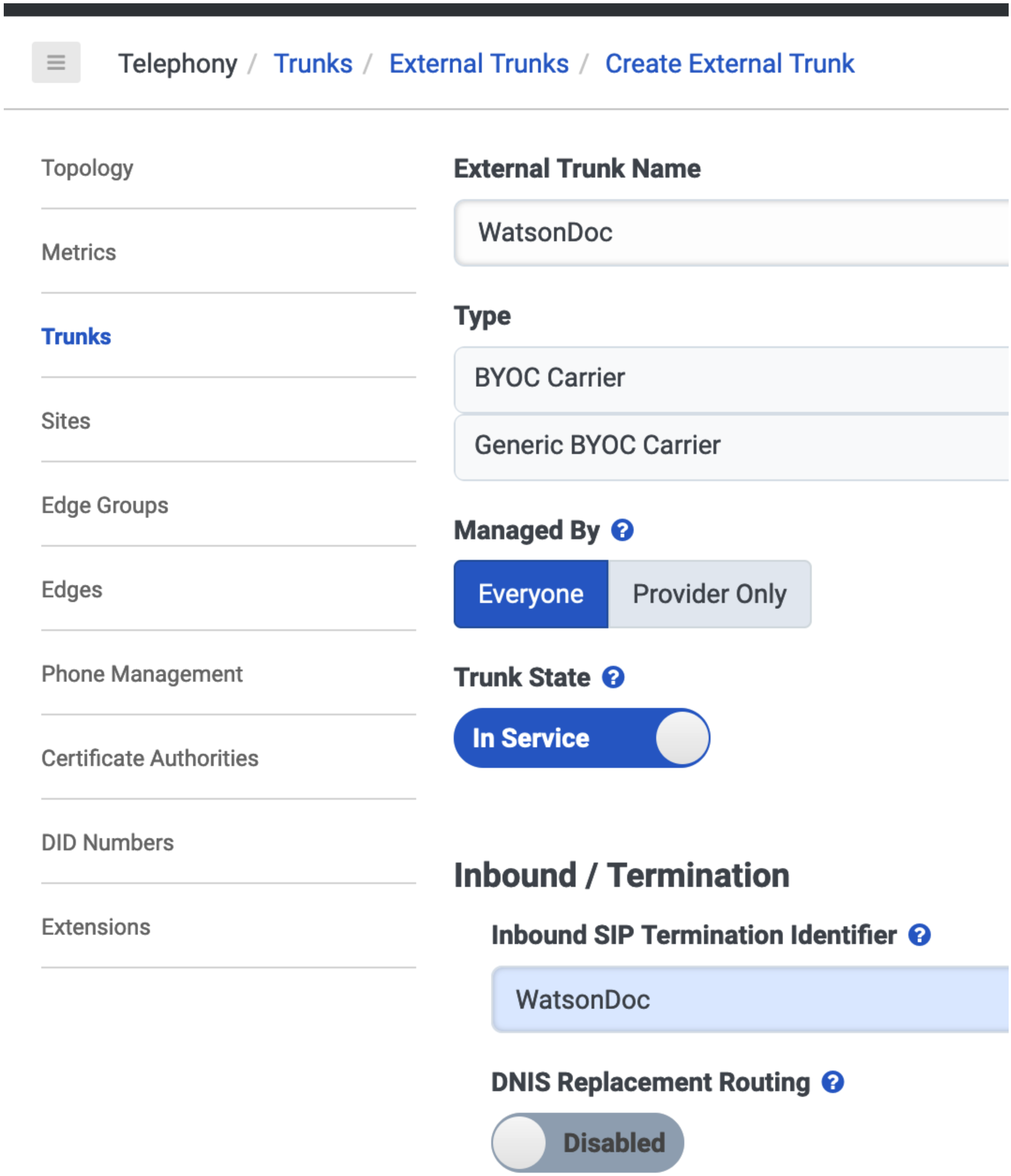
- watsonx Assistant Plus or Enterprise plan (required for phone integration).
- A working assistant that you are ready to deploy.
- A Genesys Cloud account.

Integrating with Genesys Cloud

To integrate your assistant with Genesys Cloud, follow these steps:

1. Log in to the Genesys Cloud console.

2. Click **Admin**.
3. On the **Telephony** tab, click **Trunks**.
4. In the **External Trunks** section, click **Create new**. Specify the following information:
    - In the **External Trunk Name** field, type a descriptive name (for example, `Watson`).
    - In the **Type** field, select **BYOC Carrier** and then **Generic BYOC Carrier**.
    - In the **Inbound SIP Termination Identifier** field, specify any name that you want to use (for example, `Watson`). This value is not used for now, but it is required by Genesys Cloud.
    - In the **Protocol** field, select *TLS*.



5. Under **Outbound**, scroll to the **SIP Servers or Proxies** section.
   - In the **Hostname or IP Address** field, type the SIP URI (not including `sips:`) from your watsonx Assistant phone integration settings.
   - In the **Port** field, type `5061`.

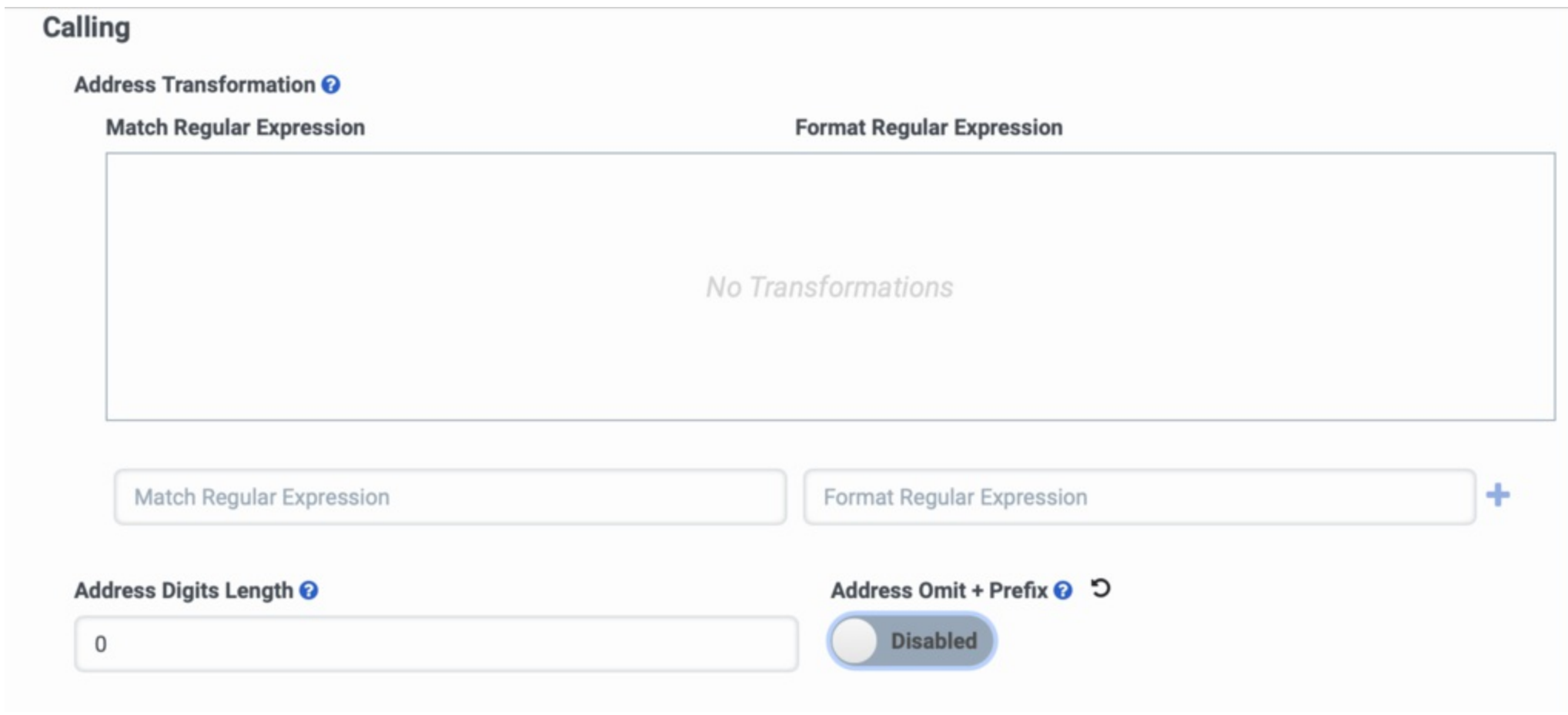
   Click the `+` button.

> **Note:** Currently, SIPS and digest authentication are supported.

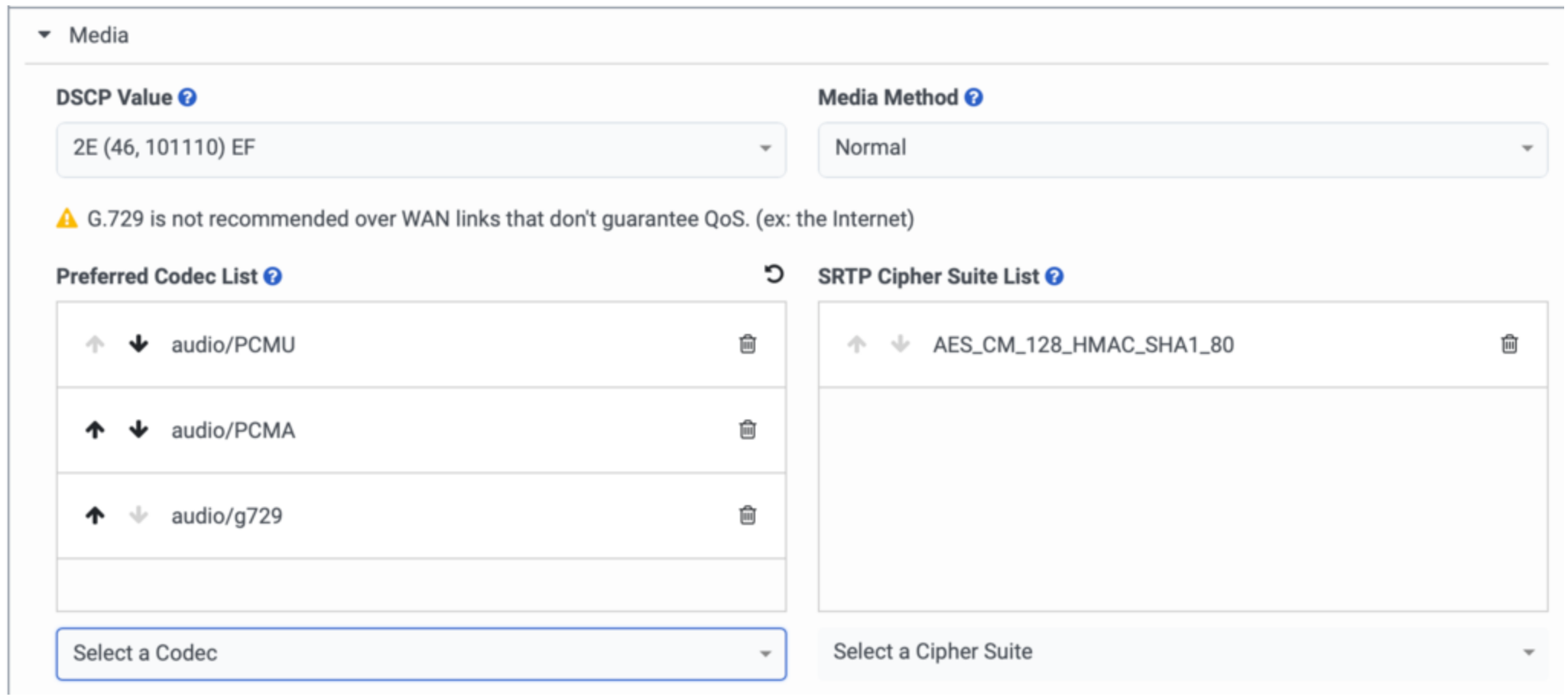
6. Under **SIP Access Control**, add the IP addresses for the data center where your assistant is located.

Data center	IP addresses
US-South	67.228.108.82 169.63.5.162 150.239.30.146
US-East	52.116.100.158 169.61.70.162 169.59.136.194
EU-DE	161.156.178.162 169.50.56.146 149.81.86.82
EU-GB	158.176.120.162 141.125.102.34 158.175.99.34
AU-SYD	168.1.47.2 135.90.86.50 168.1.106.130
JP-TOK	165.192.69.82 128.168.105.178 161.202.149.162

7. Under **Identity**, toggle the **Address Omit + Prefix** switch to **Disabled**.



8. Under **Media**, remove **Opus** from the **Preferred Codec List**. Click **Select a Codec** and then select **g729** to add it to the list. Leave **PCMU** as the first item in the list.

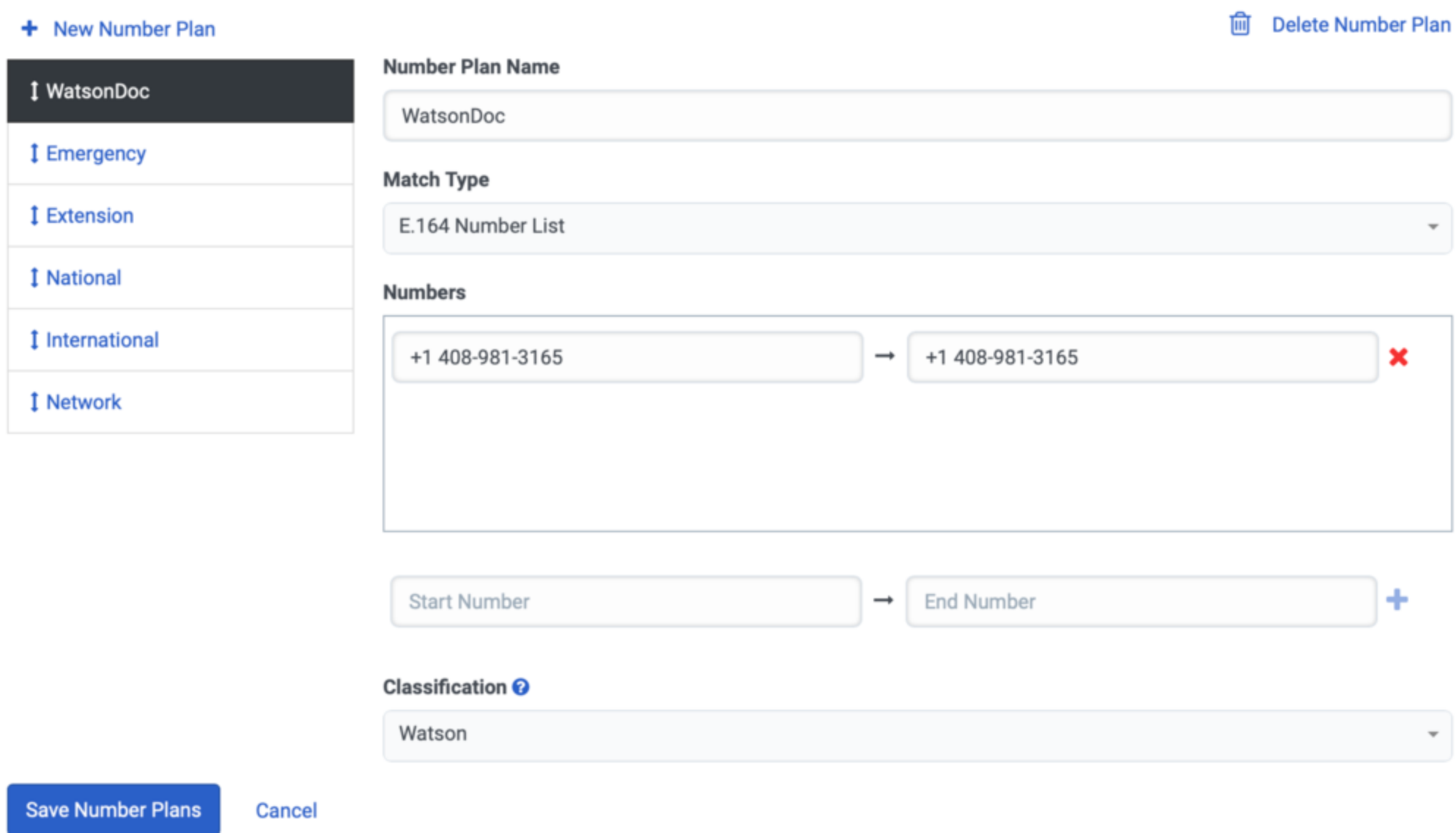


9. Under **Protocol**
    - In the **Header/Invite** section, toggle the **Conversation Headers** switch to **Enabled**.
    - Enable **Take Back and Transfer**.
10. Click **Save External Trunk**.
11. Under **Sites**, select the existing site that you want to use this trunk with. To create a new site, specify a name and location, and click **Create**.
12. Click **Number plans**. Create a number plan and specify the following information:
    - In the **Number Plan Name** field, type a descriptive name (for example, `Watson`).
    - For **Match type**, select **E. 164 Number List**.
    - In the **Numbers** field, type a number in the **Start** and **End** fields. You can make up any number to use as an identifier to assign to Watson. Specify the same number in both fields.

      Note: To create a PSTN number you can give to your clients, you must create a Direct Inward Dialing (DID) or Bring Your Own Carrier (BYOC) number. For more information, see the Genesys documentation.

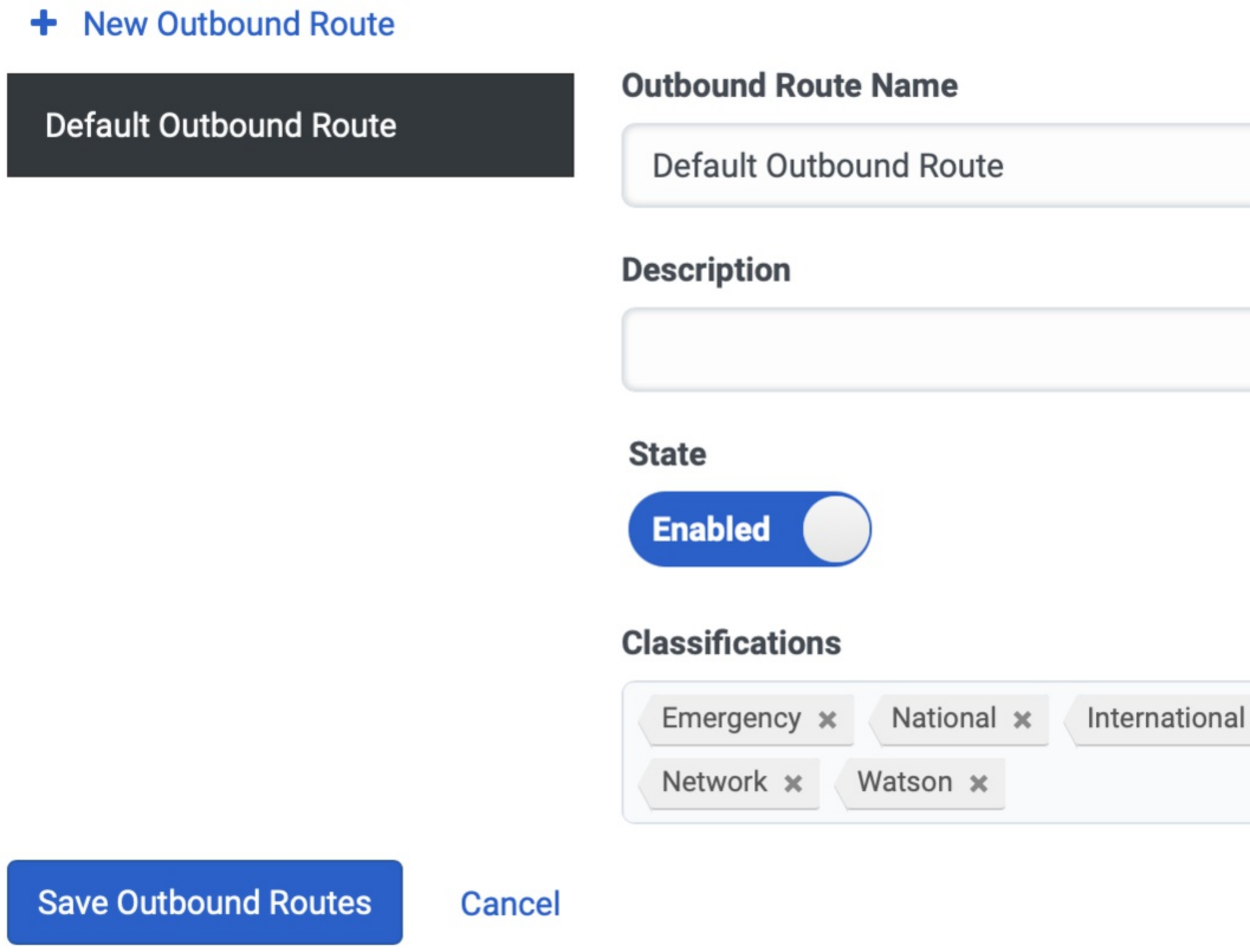
    - In the **Classification** field, type a classification name (for example, `Watson`).

    Click **Save Number Plans**.

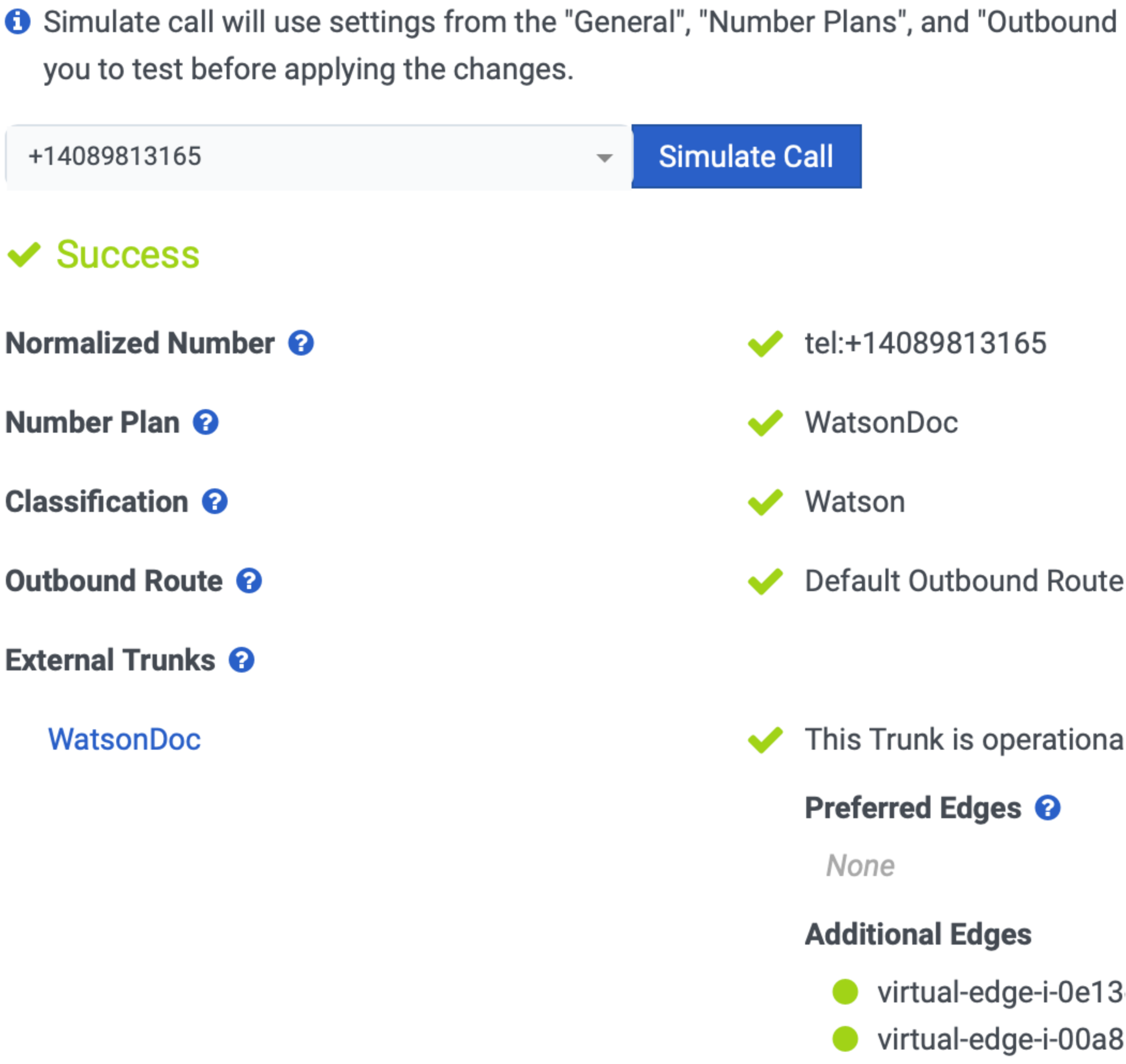


13. Click **Outbound Routes**. You can either edit the default outbound route or create a new one. Specify the following information:

    - In the **External Trunks** field, click **Select External Trunks**. Select the trunk that you created for watsonx Assistant.
    - In the **Classifications** field, add the applicable classifications. You need to include `National` and the classification you created for watsonx Assistant earlier. (The `National` route is used only to simulate the call to ensure that the trunk is operational.)
    - Toggle the **State** switch to **Enabled**.



14. Click **Save Outbound Routes**.
15. Go to the **Simulate Call** tab, and click the **Simulate Call** button. The trunk should show as operational. No actual call is made during the simulation.



16. Go to **Phone Management** and click **Create new**.
    - In the **Phone Name** field, enter a descriptive name.
    - In the **Base Settings** field, select **WebRTCPhone**.
    - In the **Site** field, select the site that you want to use.
    - In the **Person** field, select yourself.
17. In the watsonx Assistant user interface, create a new phone integration.
    - When prompted, select **Use an existing phone number with an external provider** .
    - Enter the phone number that you assigned in the Genesys **Number Plans** setting. It is not necessarily a real phone number; it is an identifier that you assigned.
    - Complete the phone integration setup process. For more information, see Integrating with phone.
    - After the phone integration is set up, go to the **SIP trunk** tab and deselect the **Don't place callers on hold while transferring to a live agent**

option.

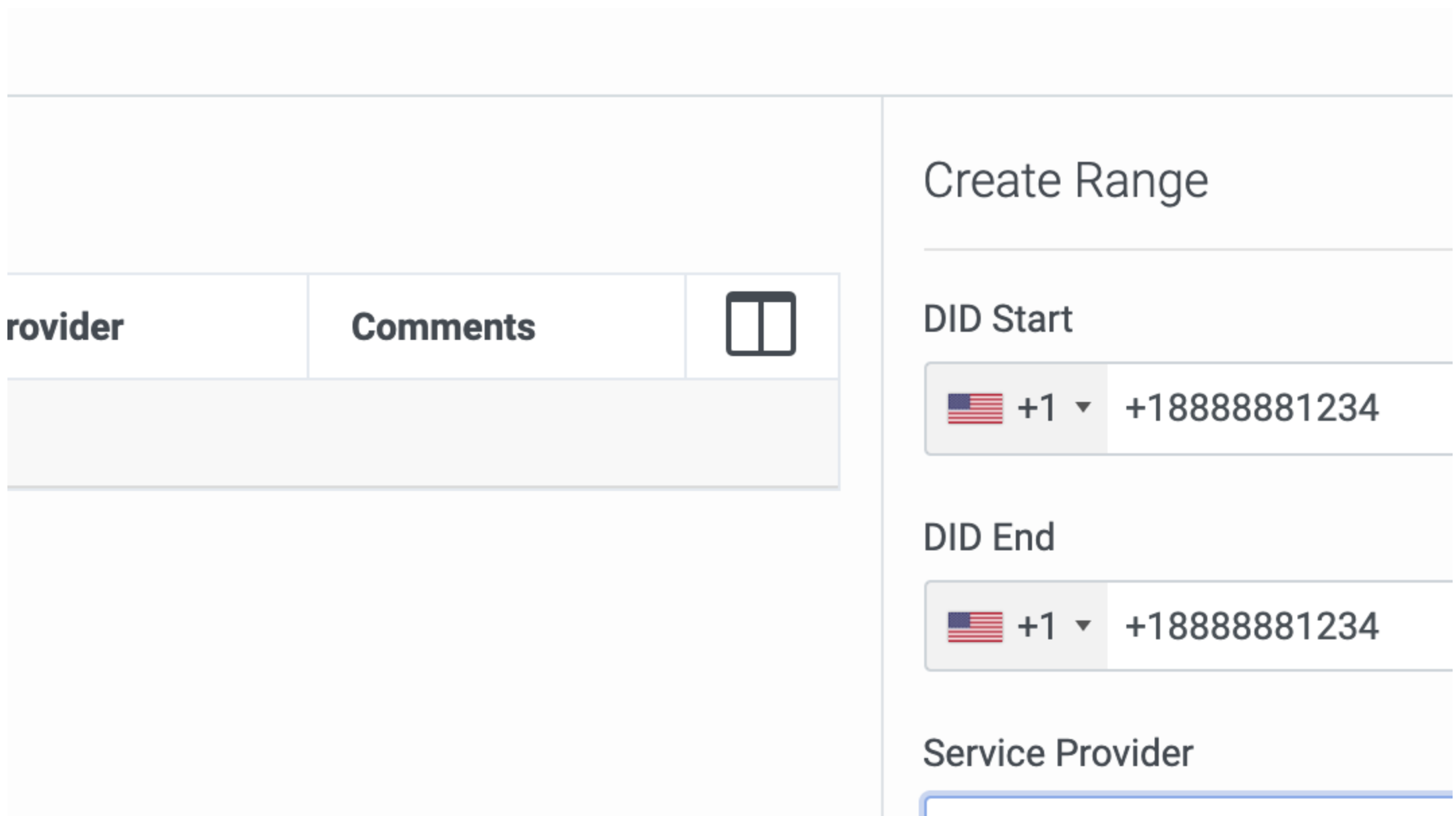
18. In the Genesys Cloud console, click the circle in the upper left. Select **Phone**, and then choose the phone that you created in the **Phone management** section. Set yourself as available. The phone icon on the left is now active.
19. Click `+` to start a new call. Specify the number that you assigned to watsonx Assistant, and then click **Dial**. You should hear your assistant speak.

> **Note:** If you encounter errors, click **Performance -> Interactions** and view the PCAP file to read the diagnostics.

Transferring to a live agent

Now that your Genesys Cloud environment can connect to watsonx Assistant, you can set up the ability for your assistant to transfer calls back to your live agents.

1. In the Genesys Cloud console, go to **DID Numbers -> DID Ranges** and create a new range.
    - In the **DID Start** and **DID End** fields, specify a phone number. (You do not need to use a real phone number; you can make up an identifier for your Genesys environment, such as `1-888-888-1234`.)

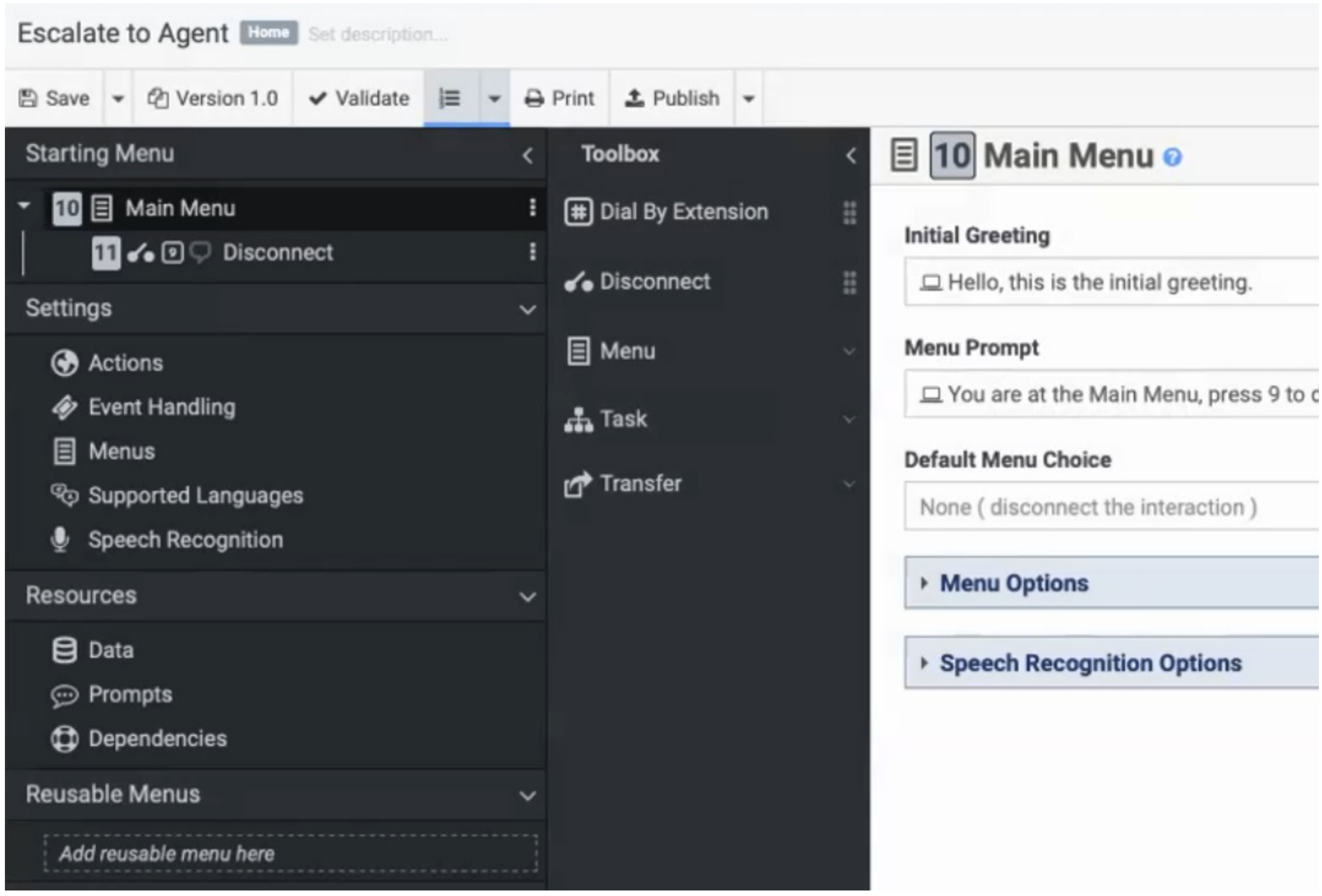


- In the **Service Provider** field, enter a descriptive name (for example, `Watson`).

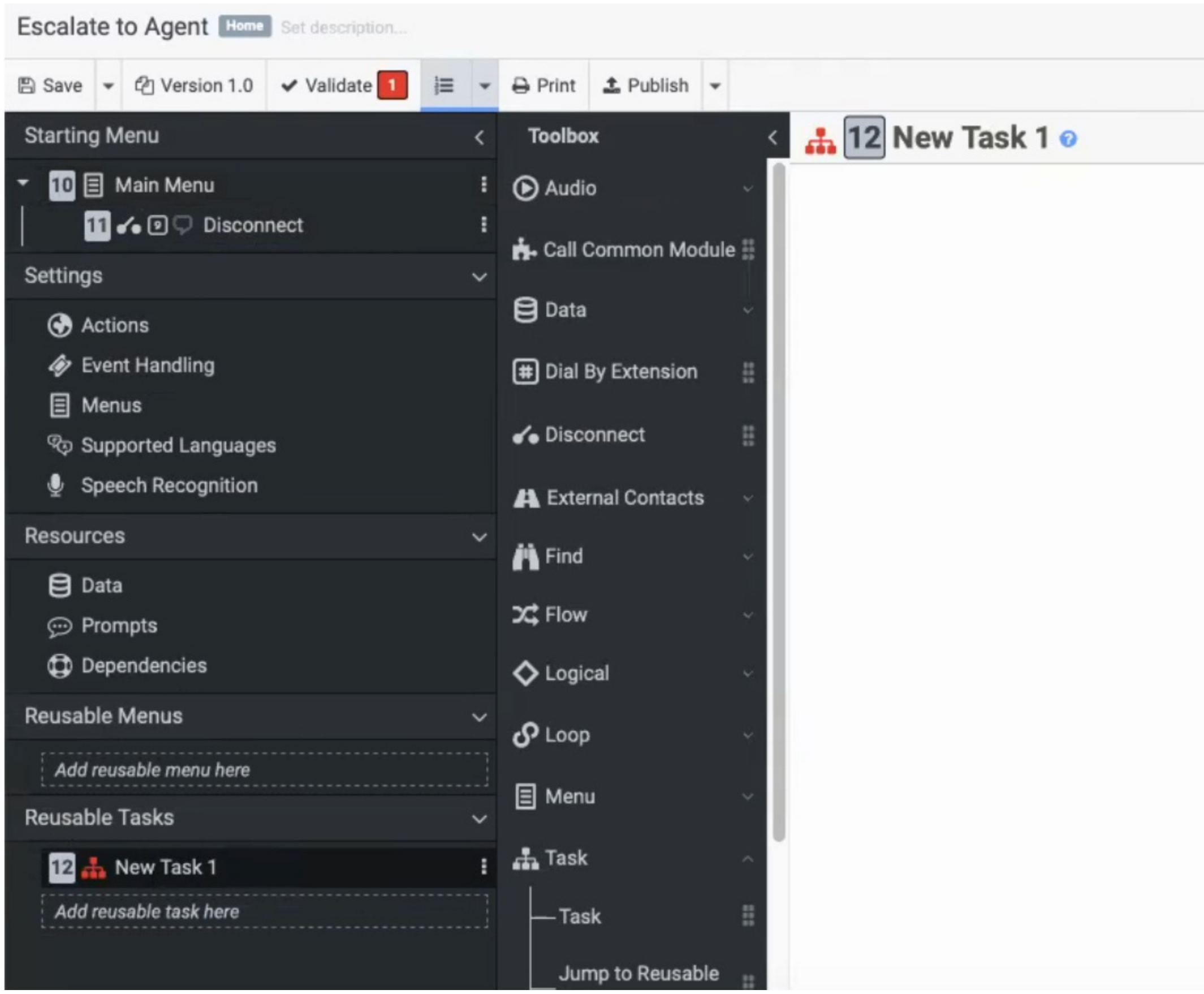
2. If you did not set up a queue to enable callers to wait for available agents, follow these steps to create one.
    a. Click **Admin**.
    b. Under **Contact Center**, click **Queues**.
    c. Create a queue, and give it a descriptive name.
    d. Add yourself as a member.
    e. Click **Save**.
3. Create a simple call flow. Your business might already have something more complex for routing.
    a. Click **Admin**.

b. Click **Architect**.

c. In the **Flows: Inbound Call** section click `+` to create a new flow. Give it a descriptive name (for example, `Escalate to Agent`).

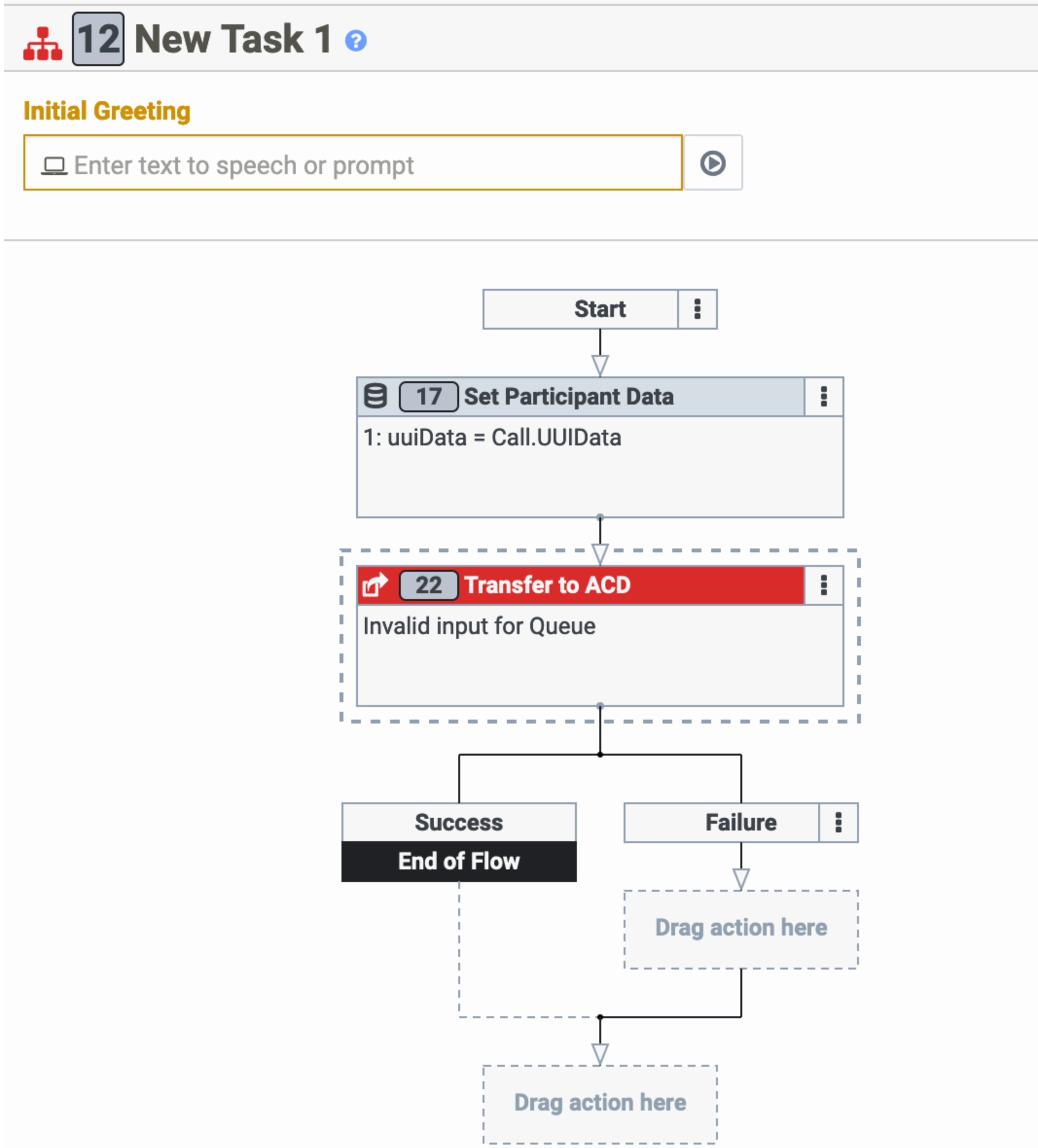


a. In the toolbox, click **Task** and drag it into **Reusable Tasks**.



a. From your toolbox under **Data**, drag a **Set Participant Data** widget into the first action. Click on the widget and specify an attribute that is named **uuiData**. For the value to assign, choose the **Expression** type and enter `Call . UUIData`.

b. From your toolbox, under **Transfer**, drag the **Transfer to ACD** widget into the first action.



a. Select the queue that you want to use.

b. In the toolbox, click the **Disconnect** widget and drag it into the action after **Failure**. (This tool disconnects the call if transfer fails.)

c. Click the three dots under **Reusable tasks** and click **Set this as the starting task**. You can remove the **Initial Greeting** because your assistant speaks before handing off the call.

d. Click **Publish** in the menu bar to make this transfer live.

e. Return to the main Genesys Cloud console.

f. Click **Admin** and then navigate to **Call Routing** in the **Routing** section.

g. Give the route a descriptive name (for example, `Escalate to Agent`).

h. Under **Regular Routing**, for all calls, select your new flow.

i. Assign the DID number that you previously created.

j. Click **Save**.

4. Make sure that your assistant is configured to transfer calls to an agent by using the *Connect To Agent* response_type. For more information, see Transferring a call to a live agent.

   For the `sip.uri` parameter, use the DID number that you created in Genesys Cloud, and the inbound SIP URI from your Genesys trunk. Use the following format:

```
$ {
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "service_desk": {
            "sip": {
              "uri": "sip:+18883334444\\@example.com",
              "transfer_headers_send_method": "refer_to_header"
            }
          }
        }
      },
      "agent_available": {
        "message": "Ok, I'm transferring you to an agent"
      },
      "agent_unavailable": {
        "message": ""
      }
    }
  ]
}
```

   Note: Make sure you use the `\\` escape characters so watsonx Assistant does not misinterpret the `@` as part of the entity shorthand syntax.

5. Make a test call and say something that initiates a transfer to an agent. In your Genesys Cloud console, you can see the transfer take place.

Share the conversation history with service desk agents

To enable the service desk agent to get a quick view of the conversation history between the visitor and the assistant, set up the watsonx Assistant Agent App app for your Genesys Pure Cloud Environment. For more information, see the Genesys starter kit.

Integrating with phone and NICE CXone contact center Plus

IBM Cloud

Connect your assistant to a NICE CXone contact center with live agents.

Transfer customers from a chat with your assistant to live agents who can help them by phone. If customers ask to speak to someone, your assistant can forward them directly to customer support with the conversation history.

This integration creates a connection between your assistant and a contact center with NICE CXone.

You need a Plus or Enterprise Plan to use this feature.

Before you begin

You must have a NICE CXone account and phone numbers that are allocated for this integration.

1. Go to the NICE website.
2. Create an account.
3. Follow the instructions to get phone numbers or select existing phone numbers.

Generate NICE CXone access keys

Access keys are used for authentication and consist of two parts: an access key ID and a secret access key.

To generate NICE CXone access keys to use with your assistant:

1. Log in to the NICE CXone console.
2. Click the app selector and select **Admin**.
3. Click **Users**, and then locate and click the user account that you want to use for the integration.
4. Click the **Access Keys** tab.
5. Click **Generate New Access Key**.
6. Click **Show Secret Key**, and copy the secret key to a secure location.

   You cannot retrieve the secret key again after you complete the next step and **Save**. You must generate a new key if the current one is lost or forgotten.
7. Click **Save**.

Set up the integration

To complete setup, you must have an assistant ready to deploy, your NICE CXone access keys, and phone numbers allocated for this integration.

To integrate your assistant with NICE CXone:

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you see a tile for **Phone**.
2. On the **Phone** tile, click **Add**.
3. On the pop-up window, click **Add** again.
4. Select **NICE CXone** on the **Select contact center** page.

   Click **Next**.
5. On the **Connect to contact center** page, specify the following values: - the **Access key ID** - the **Access key secret**.

   Click **Test connection** to verify the credentials.

   Click **Next**.
6. On the **Phone number** page, enter a phone number that you allocated for the NICE CXone integration. You can add more phone numbers later.

   Click **Next**.
7. On the **Speech to Text** page, select the instance of the Speech to Text service that you want to use.
   - If you have existing Speech to Text instances, select the instance from the list.
   - If you do not have any existing Speech to Text instances, click **Create new instance** to create a new Plus or Enterprise instance.
8. In the **Choose your Speech to Text language model** field, select the language that you want to use.

   The list of language models is automatically filtered to use the same language as your assistant. To see all language models, toggle the **Filter models based on assistant language** switch to **Off**.

   > **Note:** If you created specialized custom models that you want your assistant to use, choose the Speech to Text service instance that hosts the custom models now, and you can configure your assistant to use them later. The Speech to Text service instance must be hosted in the same location as your watsonx Assistant service instance. For more information, see Using a custom language model.

   For more information about language models, see Languages and models in the Speech to Text documentation.

   Click **Next**.
9. On the **Text to Speech** page, select the instance of the Text to Speech service that you want to use.
   - If you have existing Text to Speech instances, select the instance from the list.
   - If you do not have any existing Text to Speech instances, click **Create new instance** to create a new Standard instance.

Click **Next**.

10. On the **Contact center integrations** page, click **Test connection** near the setup information you entered.
    - If **Invalid**, check your credentials and enter each again.
    - If the credentials are correct, the **Save and exit** button becomes clickable.

    Click **Save and exit**.

The connection between your assistant and NICE CXone is complete.

> **Note:** For security reasons, the authentication fields are removed from view after initial setup. If a field required for authentication is changed, then all entries in related fields must be filled and validated again.

Configuring the NICE CXone script

NICE CXone provides a scripting tool that allows workflow developers to define routing flows for their contact centers in CXone.

The following actions and settings in the workflow are necessary for integration to work properly.

Connecting a caller to your assistant

Use the Sipputheader action. In the **headerName** property, enter the name of the SIP header field that contains the Contact ID. This header field is included in outgoing SIP INVITE messages to watsonx Assistant.

- **headerName** X-ContactID
- **headerValue** {ContactId}

Transferring a caller to a live agent

You can configure your assistant to transfer a customer to a NICE CXone live agent.

The phone integration uses the signal REST API. The **p1** attribute is preserved for the session history key. The key can be used to fetch the conversation history and present it to a live agent.

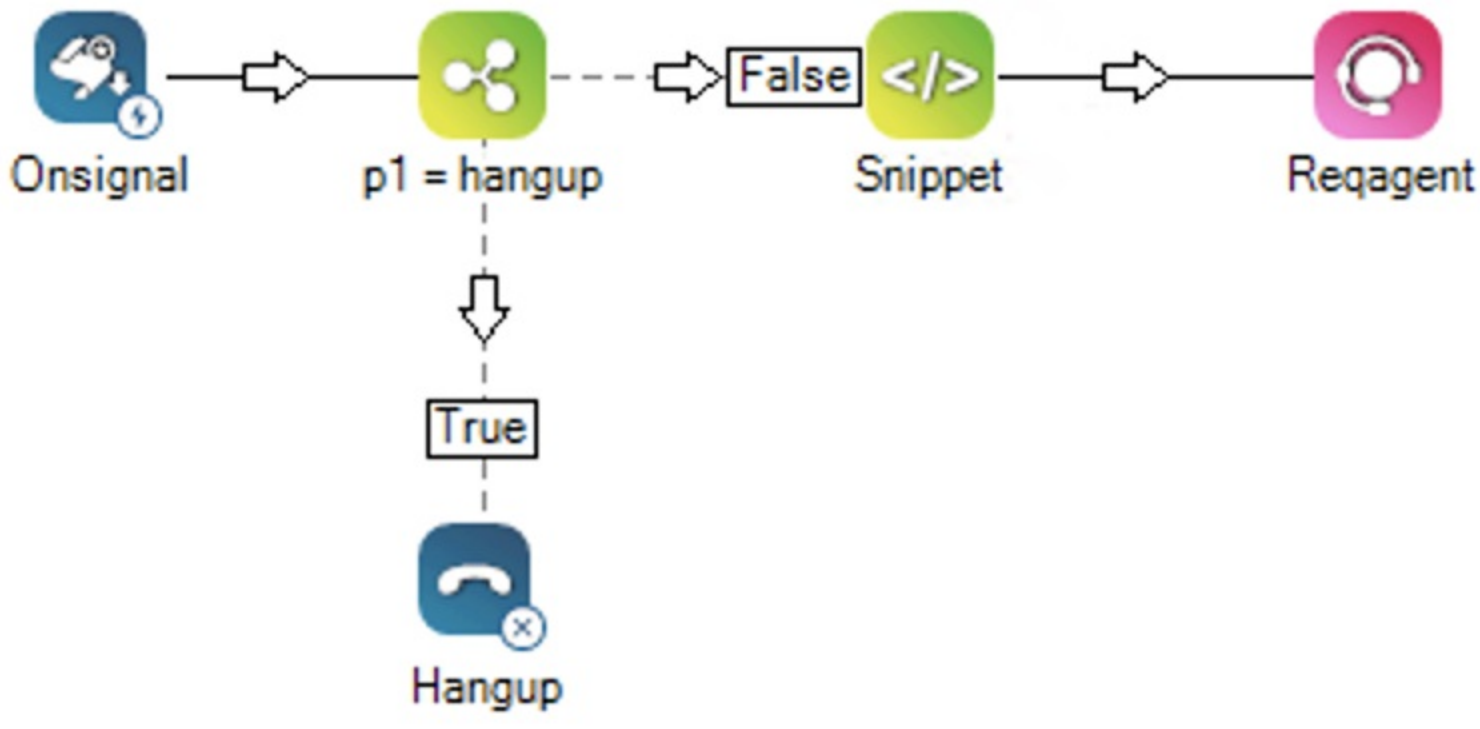
1. Use Onsignal to process the signal event.
2. Save the value of the **p1** attribute:

```
$ sessionKey = "{p1}"
```

For example, you can use the Snippet action:

```
$ ASSIGN sessionKey = "{p1}"
```

Use Reqagent to transfer a call to a live agent.



Displaying conversation history to a live agent

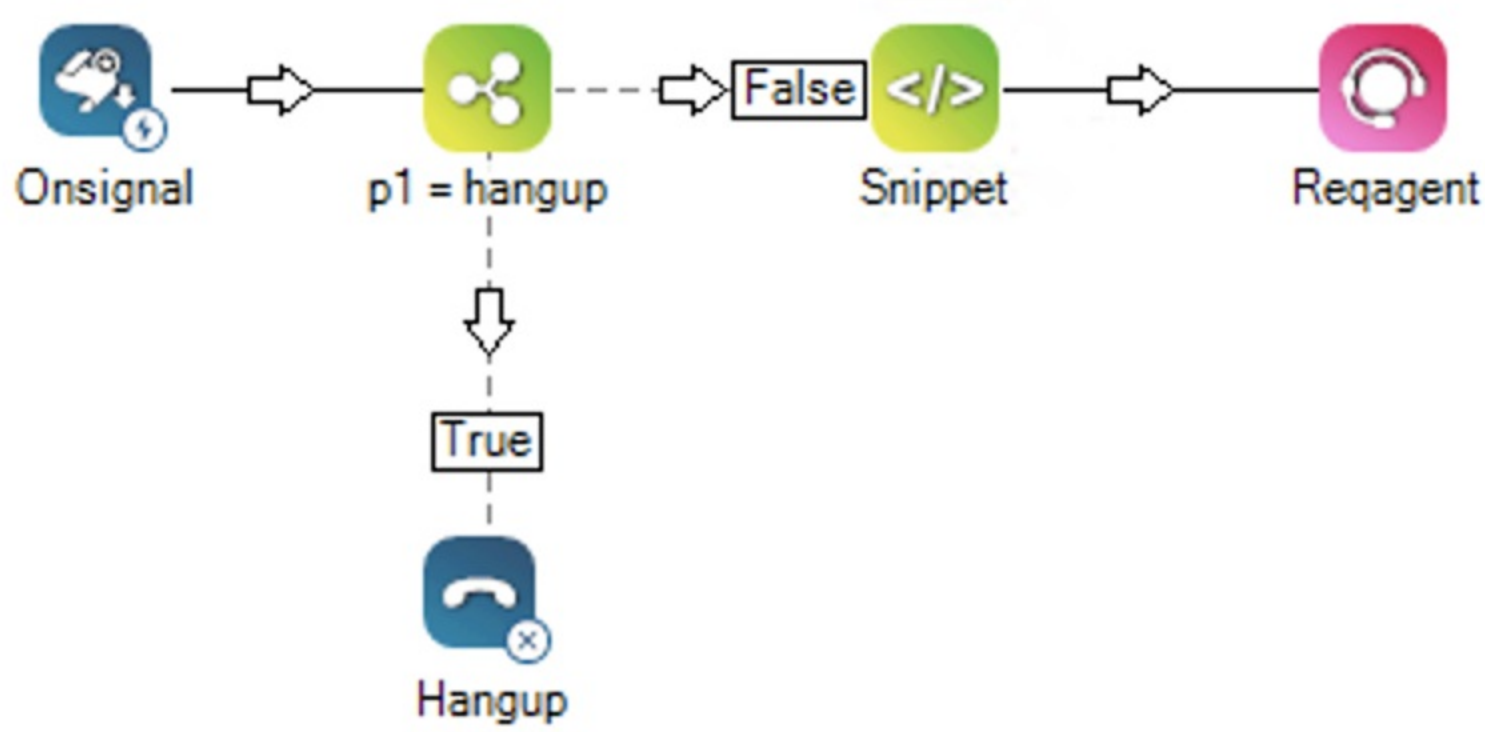
Configure your script to provide a transcript of the assistant conversation to a live agent in a pop-out window, so the agent can better understand and address a customer's needs.

1. Add an Onanswer event to your script. The **Onanswer** event is triggered when an agent answers the call.
2. Use Assign to store the link to the conversation history into a variable. **Variable** watson_url **Value** https://web-chat.global.assistant.watson.appdomain.cloud/loadAgentAppFrame.html?session_history_key={sessionKey}
3. Use the PopURL action to display the conversation transcript to a live agent. **URL** {watson_url}

Disconnecting a call

The phone integration uses the same signal REST API for disconnecting a call.

The **p1** attribute is set to `hangup`. You need to design your script so it can distinguish between transferring a call to a live agent and disconnecting a call. If **p1** is set to `hangup` when an **Onsignal** event is triggered, use Hangup to terminate a script.



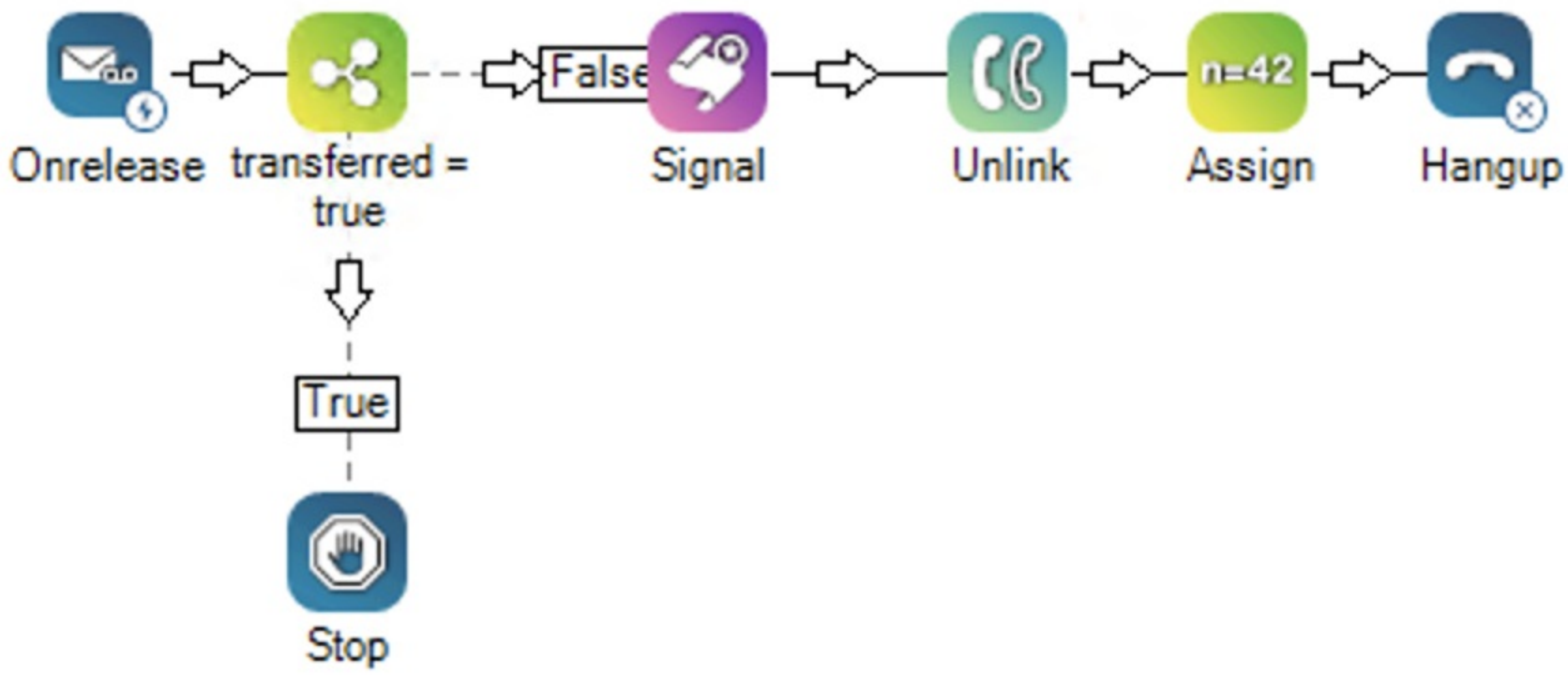
Transferring to a live agent when an error occurs

If an error occurs during a conversation, the phone integration disconnects the call by sending a `SIP BYE` request.

Use Onrelease to process the BYE request and transfer a call to a live agent.

In this example, when **Onrelease** is triggered, the script verifies whether the call was already transferred. If not, it calls the **Signal** action and sets an indication that the call is being transferred to a live agent. The indication is set using the **Assign** action.

- **Variable** transferred
- **Value** true



If the **Hangup** action is executed and an [Onrelease] event action is present, CXone hangs up on the caller, and the script jumps to the **OnRelease** action.

Design your script so it can distinguish whether the **OnRelease** event is triggered due to a transfer or hangup.

Adding transfer support to your assistant

Use the *Connect To Agent* response_type to configure your assistant to transfer calls to an agent. For instructions, see [Transferring a call to a live agent].

Use the following format:

```
{
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "nice_cxone": {
            "custom_data": {
              "p2": "test"
            }
          }
        }
      },
      "agent_available": {
        "message": "Ok, I'm transferring you to an agent."
      },
      "agent_unavailable": {
        "message": "Agent is unavailable."
      }
    }
  ]
}
```

Parameters listed in the `custom_data` object are transferred to the [signal] REST API.

Supported parameters are `p2`, `p3`, ... `p9`. `p1` is preserved and used by the phone integration for passing the session history key into the NICE CXone script.

Integrating with phone and Twilio Flex

IBM Cloud

You can use the phone integration to help your customers over the phone and transfer them to live agents inside of Twilio Flex. If, in the course of a conversation with your assistant, a customer asks to speak to a person, you can transfer the conversation directly to a Twilio Flex agent.

Before you begin

To use this integration pattern, you need:

- watsonx Assistant Plus or Enterprise Plan (required for phone integration)
- A Twilio account with the following products:
  - Twilio Flex
  - Twilio Voice with Programmable Voice API
  - Twilio Studio

Adding the watsonx Assistant phone integration

You can skip this section if you already added the phone integration to your assistant.

If you need to add the phone integration, follow these steps.

1. In the **Integrations** section on the main page for your assistant under **Essential Channels**, you will see a tile for **Phone**.
2. On the **Phone** tile, click **Add**.
3. On the pop-up window, click **Add** again.
4. Select **Use an existing phone number with an external provider** .
5. Complete the phone integration setup process. (For more information, see Integrating with phone.)

For now, this is all you need to do. For more information about configuring the phone integration, see Integrating with phone.

Adding the Twilio Flex Project

A new or an existing Twilio Flex project is required.

If you need a Twilio Flex project, you can create one using these steps.

1. From the project drop-down menu, click **Create New Project**. Specify a name for the project, and verify your account information.
2. On the welcome page, select **Flex** as the Twilio product for your new project. Complete the questionnaire, and then click **Get Started with Twilio**.

   After your Flex project is provisioned, return to the Twilio console. Make sure you selected the correct project from the drop-down list.
3. In the navigation menu, click the **All Products & Services** icon.
4. Click **Twilio Programmable Voice > Settings > General** .
5. Under **Enhanced Programmable SIP Features** , toggle the switch to **Enabled**.

Creating the call flow

After your phone integration and Twilio Flex project are configured, you must create a call-flow with Twilio Studio and provision (or port) the phone number you want your assistant to work with.

To create the call flow:

1. In the navigation menu, click the **All Products & Services** icon.
2. Click **Studio**.
3. Click **+** to create a new flow.
4. Name the new flow, and then click **Next**.
5. Select **Start From Scratch**, and then click **Next**. A **Trigger** widget appears in your flow canvas.
6. Click the **Trigger** widget.
7. Make note of the value from the **WEBHOOK URL** field. You need this value in a subsequent step.

Configuring the phone number

1. In the navigation menu, click the **All Products & Services** icon.
2. Click **Phone Numbers**.
3. Under **Manage Numbers**, configure the phone number you want your assistant to use. Select **Buy a Number** to buy a new number, or **Port & Host** to port an existing phone number.

4. In the **Active Numbers** list, click the new phone number.
5. Under **Voice and Fax**, configure these settings.
    - For **CONFIGURE WITH** field, select **Webhook, TwiML Bins, Functions, Studio, or Proxy**.
    - For **A CALL COMES IN**, select **Studio Flow**. Select your flow from the drop-down list.
    - For **PRIMARY HANDLER FAILS**, select **Studio Flow**. Select your flow from the drop-down list.
6. Go to the watsonx Assistant user interface, and open the phone integration settings for your assistant.
7. In the **Phone number** field, type the phone number that you configured in Flex Studio.
8. Click **Save and exit**.

Test your phone number

You can now test that your phone number is connected to your flow by triggering a **Say/Play** widget in the Twilio Flex Flow editor.

1. Drag a **Say/Play** widget onto your flow canvas.
2. Configure the Say/Play widget with a simple phrase like `I'm alive.`.
3. Connect the **Incoming call** node on your **Trigger** widget to your **Say/Play** widget.
4. Call your phone number. You should hear Twilio flow respond with your test phrase.
5. Delete the **Say/Play** widget and continue to the next step.
6. If this test did not work as expected, double-check your phone number configuration to ensure it is attached to your flow.

Creating a Twilio function to handle incoming calls

Now you need to configure the call-flow to direct inbound calls to the assistant by using a Twilio function. Follow these steps:

1. In the navigation menu, click the **All Products & Services** icon.
2. Click **Services**.
3. Click **Create Service**. Specify a service name and then click **Next**.
4. Click **Add** > **Add Function** to add a new function to your service. Name the new function `/receive-call`.
5. Replace the template in your `/receive-call` function with this code:

```
$ exports.handler = function(context, event, callback) {
  const VoiceResponse = require('twilio').twiml.VoiceResponse;
  const response = new VoiceResponse();
  const dial = response.dial({
    answerOnBridge: "true",
    referUrl: "/refer-handler"
  });
  const calledPhoneNumber = event.Called;
  dial.sip(`sip:${calledPhoneNumber}@{sip_uri_hostname};secure=true`);
  return callback(null, response);
}
```

    - Replace `{sip_uri_hostname}` with the hostname portion of your assistant's phone integration SIP URI (everything that comes after `sips:`). Note: Twilio does not support `SIPS` URIs, but does support secure SIP trunking by appending `;secure=true` to the SIP URI.
6. Click **Save**.
7. Click **Deploy All**.

Redirecting to the incoming call handler

Use a TwiML **Redirect** widget in your Studio Flow editor to call out to the `/receive-call` function created in the previous section.

1. Add a **TwiML Redirect** widget to your Studio Flow canvas.
2. Connect the Incoming Call trigger to your **TwiML Redirect** widget.

3. Configure the **TwiML Redirect** widget with the URL for the `/receive-call` function that you created in the previous section.
4. Your flow should now redirect to watsonx Assistant when receiving an inbound call.
5. If the redirect fails, make sure you deployed your `/receive-call` function.

Creating a Twilio function to handle transfers from assistant

You also need to configure the call-flow to handle calls transferred from the assistant back to Twilio Flex, for cases when customers ask to speak to an agent. Use a **Say/Play** after the **TwiML Redirect** widget to show that the call is transferred back to the flow from watsonx Assistant. Many options are possible at this point, such as queuing the call for a live agent, and are discussed in this section.

1. Add a **Say/Play** widget to your canvas, and configure it with a phrase such as, `Transfer from Watson complete`.
2. Connect the **Return** node on the **TwiML Redirect** widget to your **Say/Play** widget.
3. Click the **Trigger** widget.
4. Copy the value from the **WEBHOOK URL** field. You need this value in a subsequent step.
5. On the Twilio Functions page, click **Add** > **Add Function** to add another new function to your service. Name this new function `/refer-handler`.
6. Replace the template in your `/refer-handler` function with the following code:

```
$ exports.handler = function(context, event, callback) {
  // This function handler will handle the SIP REFER back from the Phone Integration.
  // Before handing the call back to Twilio, it will extract the session history key from the
  // User-to-User header that's part of the SIP REFER Refer-To header. This session history key
  // is a string that is used to load the agent application in order to share the transcripts of the caller
  // with the agent.
  // See https://github.com/watson-developer-cloud/assistant-web-chat-service-desk-starter/blob/main/docs/AGENT_APP.md
  const VoiceResponse = require('twilio').twiml.VoiceResponse;

  const STUDIO_WEBHOOK_URL = '{webhook_url}';

  let studioWebhookReturnUrl = `${STUDIO_WEBHOOK_URL}?FlowEvent=return`;

  const response = new VoiceResponse();
  console.log("ReferTransferTarget: " + event.ReferTransferTarget);

  const referToSipUriHeaders = event.ReferTransferTarget.split("?")[1];
  console.log(referToSipUriHeaders);
  if (referToSipUriHeaders) {
    const sanitizedReferToSipUriHeaders = referToSipUriHeaders.replace(">", "");
    console.log("Custom Headers: " + sanitizedReferToSipUriHeaders);

    const sipHeadersList = sanitizedReferToSipUriHeaders.split("&");

    const sipHeaders = {};
    for (const sipHeaderSet of sipHeadersList) {
      const [name, value] = sipHeaderSet.split('=');
      sipHeaders[name] = value;
    }

    const USER_TO_USER_HEADER = 'User-to-User';

    // Extracts the User-to-User header value
    const uuiData = sipHeaders[USER_TO_USER_HEADER];

    if (uuiData) {
      const decodedUUIData = decodeURIComponent(uuiData);
      const sessionHistoryKey = decodedUUIData.split(';')[0];
      // Passes the session history key back to Twilio Studio through a query parameter.
      studioWebhookReturnUrl = `${studioWebhookReturnUrl}&SessionHistoryKey=${sessionHistoryKey}`;
    }
  }

  response.redirect(
    { method: 'POST' },
    studioWebhookReturnUrl
  );
```

```
  // This callback is what is returned in response to this function being invoked.
  // It's really important! E.g. you might respond with TWiML here for a voice or SMS response.
  // Or you might return JSON data to a studio flow. Don't forget it!
  return callback(null, response);
}
```

Replace `{webhook_url}` with the **WEBHOOK URL** value that you copied from the **Trigger** widget in your Studio Flow.

7. Click **Save**.
8. Click **Deploy All**.
9. After you create this refer-handler, copy the function URL back into the `/receive-call` handler's **referUrl** field.

Configuring the assistant to transfer calls to Twilio Flex

Now we need to configure the assistant to transfer calls to Twilio Flex when a customer asks to speak to an agent. Follow these steps:

1. In the watsonx Assistant user interface, open the dialog skill of your assistant.
2. Add a node with the condition that triggers your assistant to transfer customers to an agent.
3. Add a text response to the node, and specify the text that you want your assistant to say before it transfers customers to an agent.
4. Open the JSON editor for the response.
5. In the JSON editor, add a `connect_to_agent` response, specifying your phone number as the `sip.uri` (replace `{phone_number}` with the phone number of your SIP trunk):

```
$ {
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
        "target": {
          "service_desk": {
            "sip": {
              "uri": "sip:+{phone_number}@flex.twilio.com",
              "transfer_headers_send_method": "refer_to_header"
            }
          }
        }
      },
      "agent_available": {
        "message": "Ok, I'm transferring you to an agent"
      },
      "agent_unavailable": {
        "message": ""
      }
    }
  ]
}
```

This example does not show how to use the context that is passed from watsonx Assistant to Twilio Flex. You can reference the User-to-User information from within the Twilio Flex flow:

```
$ {
  "context": {
    "widgets": {
      "redirect_1": {
        "User-to-User": "value",
      }
    }
  }
}
```

where `redirect_1` is the name of your redirect widget. For example, if you set up multiple queues, you might want to use a Twilio Split widget to pick a queue based on the returned context.

Test your assistant

Your assistant should now be able to answer calls to your phone number and transfer calls back to your Twilio Flex flow. To test your assistant:

1. Call your phone number. When the assistant responds, ask for an agent.
2. You should hear the phrase configured in the **Say/Play** widget (such as, `Transfer from Watson complete`).
3. If the transfer fails, use the console log to follow the flow of the call as it moves from the flow to the `/receive-call` handler, to watsonx Assistant, to the refer-handler, and then back to your Twilio Flex flow.

Share the conversation history with service desk agents

To enable service desk agents to get a quick view of the conversation history between visitor and assistant, set up the watsonx Assistant Agent App app for your Twilio Flex environment. For more information, see documentation at the Twilio Flex watsonx Assistant Agent App.

Handling phone interactions

If your assistant uses the phone integration, you can use various response types to customize the behavior of the integration or manage the flow of conversations that your assistant has with customers over the telephone.

You can use response types to perform the following phone-specific actions:

- Apply advanced settings to the Speech to Text service.
- Apply advanced settings to the Text to Speech service.
- Transfer a call to a live agent.
- Play hold music or a voice recording.
- Enable keypad entry.
- Transfer the conversation to the web chat integration.
- End the call.
- Send a text message during a phone conversation.

In some cases, you might want to combine response types to perform multiple actions. For example, you might want to implement two-factor authentication by requesting phone keypad entry and sending a text message from the same action step. For more information, see Define a sequence of phone commands.

You can also perform the following phone-specific actions:

- Inject custom values into CDR log events.
- Access phone integration context variables from your action.

For more information, see Response types reference.

Adding phone-specific responses to your assistant

To initiate a voice-specific interaction from an action step, add a response in the `generic` array with the appropriate response type. For more information about using the JSON editor to add responses, see Defining responses with the JSON editor.

Applying advanced settings to the Speech to Text service

Use the `speech_to_text` response type to send configuration commands to the Speech to Text service instance used by the phone integration. By sending a `speech_to_text` response from an action step, you can dynamically change the Speech to Text configuration during a conversation.

By default, any Speech to Text configuration changes you make persist for the remainder of the conversation, or until you update them again. You can change this behavior by specifying the `update_strategy` property of the `parameters` object.

The format of the `speech_to_text` response type is as follows:

```
{
  "generic": [
    {
      "response_type": "speech_to_text",
      "command_info": {
        "type": "<command type>",
        "parameters": {
          "parameter name1": "parameter value",
```

```
        "parameter name2": "parameter value"
      }
    }
  }
 ]
}
```

Each command type along with its related parameters are described in the following sections.

command_info.type : configure

Dynamically reconfigures the Speech to Text service by applying a set of configuration parameters, which can be based on the conversation flow. For example, you might want to choose a particular customization ID or grammar at a specific point in the conversation.

parameter	description	required	default
narrowband_recognize	The Speech to Text service configuration to use for narrowband codecs (such as PCMU and PCMA, which are sampled at 8 kHz). The parameters for this object are used when connecting to the Speech to Text service for speech recognition requests. For more information about these parameters, see the Speech to Text API documentation.	no	Current Speech to Text configuration
broadband_recognize	The Speech to Text service configuration to use for broadband codecs (such as G722, which is sampled at 8 kHz). The parameters for this object are used when connecting to the Speech to Text service for speech recognition requests. For more information about these parameters, see the Speech to Text API documentation.	no	Current Speech to Text configuration
band_preference	Specifies which audio band (narrowband or broadband) is preferred when negotiating audio codecs for the session. Set to broadband to use broadband audio when possible.	no	narrowband
update_strategy	Specifies the update strategy to use when setting the speech configuration. Possible values include: • replace: Replaces the configuration for the rest of the session. Any root-level fields in the new configuration completely overwrite the previous configuration. • replace_once: Replaces the configuration only for the next turn of the conversation. Then, the previous configuration is used. • merge: Merges the new configuration with the existing configuration for the rest of the session. Only changed parameters are overwritten; any other configuration parameters are unchanged. • merge_once: Merges the new configuration with the existing configuration only for the next turn of the conversation. Then, the previous configuration is used.	no	replace

The parameters that you can set for narrowband_recognize and broadband_recognize reflect the parameters that are made available by the Speech to Text WebSocket interface. The WebSocket API sends two types of parameters: query parameters, which are sent when the phone integration connects to the service, and message parameters, which are sent as part of the JSON data in the request body. For example, model is a query parameter, and smart_formatting is a WebSocket message parameter. For a full list of parameters, see the Speech to Text API documentation.

You can define the following query parameters for the phone integration's connection to the Speech to Text service. Any other parameter that you define for narrowband or broadband is passed through as part of the WebSocket message request.

- model
- acoustic_customization_id
- version
- x-watson-learning-opt-out
- base_model_version
- language_customization_id

The following parameters from the Speech to Text service cannot be modified because they have fixed values that are used by the phone integration.

- action
- content-type
- interim_results

- `continuous`
- `inactivity_timeout`

> **Note:** On configuring dynamically from watsonx Assistant with the `configure` command, only the root level fields, such as `narrowband` or `broadband`, are updated. If these fields are omitted from the command, the original configuration settings persist. You can use the `update_strategy` values `merge` and `merge_once` to merge configuration parameters with the existing configuration.

Using a custom language model

When you set up the phone integration, you can configure the integration to use a custom language model all the time.

However, you might want to use a standard language model most of the time, and specify a custom language model to use only for specific topics that your assistant is designed to help customers with. For example, you might want to use a custom model that specializes in medical terms for an action that helps with medical bills only. You can apply a custom language model for a specific branch of the conversation.

For more information, see Creating a custom language model.

To apply a custom language model to an action step, use the `speech_to_text` response type.

```
{
 "generic": [
  {
   "response_type": "speech_to_text",
   "command_info": {
    "type": "configure",
    "parameters": {
     "narrowband_recognize": {
      "x-watson-learning-opt-out": true,
      "model": "en-US_NarrowbandModel",
      "profanity_filter": true,
      "smart_formatting": true,
      "language_customization_id": "81d3630-ba58-11e7-aa4b-41bcd3f6f24d",
      "acoustic_customization_id": "e4766090-ba51-11e7-be33-99bd3ac8fa93"
     }
    }
   }
  }
 ]
}
```

You can also apply an acoustic model that you might train to handle background noise, accents, or other things that are associated with the quality or noise of the signal.

Using custom grammar

The Speech to Text service supports the use of grammar. You can use grammar to configure the audio to match specific characteristics only.

- A custom language model expands the service's base vocabulary.
- A grammar restricts the words that the service can recognize from that vocabulary.

When you use a grammar with a custom language model for speech recognition, the service can recognize only words, phrases, and strings that are recognized by the grammar. For example, maybe you want to accept only a `yes` or `no` response. You can define a grammar that allows only those options.

For more information, see Using grammar with custom language models.

This example shows how to specify a custom grammar during the conversation:

```
{
 "generic": [
  {
   "response_type": "speech_to_text",
   "command_info": {
    "type": "configure",
    "parameters": {
     "update_strategy": "merge_once",
     "narrowband_recognize": {
      "x-watson-learning-opt-out": true,
      "grammar_name": "names-abnf",
```

```
        "language_customization_id": "81d3630-ba58-11e7-aa4b-41bcd3f6f24d"
       }
      }
     }
    }
  ]
}
```

Examples

The following examples illustrate how to use the `speech_to_text` response type to send configuration commands to the Speech to Text service.

Example: Setting the language model

In this example, the language model is switched to Spanish (`es-ES_NarrowbandModel`), and smart formatting is enabled.

```
{
  "generic": [
    {
      "response_type": "speech_to_text",
      "command_info": {
        "type": "configure",
        "parameters": {
          "narrowband_recognize": {
            "model": "es-ES_NarrowbandModel",
            "smart_formatting": true
          }
        }
      }
    }
  ]
}
```

Example: Updating the recognizeBody property for one conversation turn

The following example shows how to specify the use of a custom language model for a single turn of the conversation turn. Set `update_strategy` to `merge_once` and specify the ID of the custom language model in the configuration parameters.

```
{
  "generic": [
    {
      "response_type": "speech_to_text",
      "command_info": {
        "type": "configure",
        "parameters": {
          "update_strategy": "merge_once",
          "narrowband_recognize": {
            "language_customization_id": "ao45vohgFuxyOQRgztu-02I10ut7aJcM-AdInT-VWgj3V"
          }
        }
      }
    }
  ]
}
```

Applying advanced settings to the Text to Speech service

Use the `text_to_speech` response type to send configuration commands to the Text to Speech service instance used by the phone integration. By sending a `text_to_speech` response from an action step, you can dynamically change the Text to Speech configuration during a conversation.

By default, any Text to Speech configuration changes you make persist for the remainder of the conversation, or until you update them again. You can change this behavior by specifying the `update_strategy` property of the `parameters` object.

The format of the `speech_to_text` response type is as follows:

```
$
```

```
{
  "generic": [
    {
      "response_type": "text_to_speech",
      "command_info": {
        "type": "<command type>",
        "parameters": {
          "parameter name": "parameter value",
          "parameter name": "parameter value"
        }
      }
    }
  ]
}
```

Each command type along with its related parameters are described in the following sections.

command_info.type : configure

Dynamically reconfigures the Text to Speech service by applying a set of configuration parameters, which can be based on the conversation flow. For example, you might want to choose a particular voice at a specific point in the conversation.

parameter	description	required	default
synthesize	The Text to Speech service configuration to use when synthesizing audio. The parameters that are defined by this object are used when connecting to the Text to Speech service for speech synthesis requests. For more information about these parameters, see the Text to Speech API documentation.	yes	Current Text to Speech configuration
update_strategy	Specifies the update strategy to use when setting the speech configuration. Possible values include: • replace: Replaces the configuration for the rest of the session. Any root-level fields in the new configuration completely overwrite the previous configuration. • replace_once: Replaces the configuration only for the next turn of the conversation. Then, the previous configuration is used. • merge: Merges the new configuration with the existing configuration for the rest of the session. Only changed parameters are overwritten; any other configuration parameters are unchanged. • merge_once: Merges the new configuration with the existing configuration only for the next turn of the conversation. Then, the previous configuration is used.	no	replace

The parameters that you can set for synthesize reflect the parameters that are made available by the Text to Speech WebSocket interface. The WebSocket API sends two types of parameters: query parameters, which are sent when phone integration connects to the service, and message parameters, which are sent as part of the JSON data in the request body. For a full list of parameters, see the Text to Speech API documentation.

command_info.type : disable_barge_in

Disables speech barge-in so that playback isn't interrupted when the caller speaks while audio is being played back.

No parameters.

command_info.type : enable_barge_in

Enables speech barge-in so that callers can interrupt playback by speaking.

No parameters.

Changing the assistant's voice

You can change the voice of your assistant when it covers certain topics in the conversation that warrant it. For example, you might want to use a voice with a British accent for a branch of the conversation that applies only to customers in the UK.

This example shows how to specify a voice during the conversation:

```
{
  "generic": [
    {
      "response_type": "text_to_speech",
      "command_info": {
        "type": "configure",
        "parameters": {
          "synthesize": {
            "voice": "en-GB_KateV3Voice"
          }
        }
      }
    }
  ]
}
```

In the `voice` parameter, specify the voice model that you want to use. For more information about voice model options, see Supported languages and voices.

Note: The model that you specify must be one that is supported by the Text to Speech service instance that is configured for use with the integration.

Transferring a call to a live agent

When you configure the phone integration, you can optionally set up backup call center support, which makes it possible for the assistant to transfer a call to a human. You can use the *Connect to agent* response type in an action step to initiate a transfer to a live agent at a specific point in the conversation. When a *Connect to agent* response is sent to the phone integration, a SIP transfer is initiated with the SIP `REFER` message, as defined by RFC 5589.

For more information about initiating a transfer to a live agent during the conversation, see the following documentation:

- Setting up live agent escalation
- Defining responses with the JSON editor

The phone integration supports more parameters for the *Connect to agent* response type. You can add these phone-specific parameters to the `connect_to_agent` response type with the JSON editor.

The `connect_to_agent` response type supports the ability to specify the target transfer information under the `transfer_info` parameter.

The following example shows a transfer that uses all of the configurable parameters:

```
{
  "generic": [
    {
      "response_type": "connect_to_agent",
      "transfer_info": {
  "target": {
    "service_desk": {
      "sip": {
        "uri": "sip:user\\@domain.com",
        "transfer_headers": [
          {
            "name": "Customer-Header1",
            "value": "Some-Custom-Info"
          },
          {
            "name": "User-to-User",
            "value": "XXXXXX"
          }
        ],
        "transfer_headers_send_method": "refer_to_header"
      }
    }
  }
},
"agent_available": {
  "message": "I'll transfer you to an agent"
},
```

```
    "agent_unavailable": {
      "message": "Sorry, I could not find an agent."
    },
    "message_to_human_agent": "The caller needs help resetting their password"
  }
 ]
}
```

The `connect_to_agent` response type supports the following phone-specific properties.

Parameter	Default	Description
`service_desk.sip.uri`	N/A	The SIP or telephone URI to transfer the call to, such as `sip:12345556789\@myhost.com` or `tel:+18883334444`. Optional for the `hangup` method.
`service_desk.sip.transfer_method`	`refer`	Determines how to transfer the call: • `refer`: The call is transferred by sending a SIP `REFER` request, which is the default value. • `hangup`: The call is transferred by sending a SIP `BYE` request.
`service_desk.sip.transfer_target_header`	`Transfer-Target`	The SIP header that contains the transfer target when a `BYE` request is used for transferring the call. This option is supported only in the `hangup` method.
`service_desk.sip.transfer_headers`	N/A	A list of custom header field name-value pairs to be added to a transfer request.
`service_desk.sip.transfer_headers_send_method`	`custom_header`	The method by which the SIP transfer headers are sent. • `custom_header`: Sends the transfer headers as part of the SIP message, which is the default value. • `contact_header`: Sends the transfer headers in the `Contact` header. This option is not supported in the `hangup` method. • `refer_to_header`: Sends the transfer headers in the `Refer-To` header. This option is not supported in the `hangup` method.

If you define a SIP URI as the transfer target, escape the at sign (`@`) in the URI by adding two backslashes (`\\`) in front of it. This prevents the string from being recognized as part of the entity shorthand syntax.

```
"uri": "sip:12345556789\\@myhost.com"
```

Transferring after hangup

By default, the phone integration transfers calls by using a SIP `REFER` request. Depending on the IVR service provider, you might need to configure call transfer to use a SIP `BYE` request instead. Use the `transfer_method` attribute to specify how to transfer the call, using either `refer` or `hangup`. When `transfer_method` is set to `hangup` instead of `refer`, the behavior of the transfer action changes. Instead of sending a SIP `REFER` request, the phone integration plays back any associated text and then hangs up the call by sending a SIP `BYE` request.

After the hangup, the phone integration passes the transfer destination that is specified in the `url` attribute to the call anchor in the `BYE` message. The header field that contains the transfer target is determined by the `transfer_target_header` attribute. If the `transfer_target_header` attribute isn't specified, the phone integration uses `Transfer-Target`.

```
{
 "generic": [
  {
   "response_type": "connect_to_agent",
   "transfer_info": {
    "target": {
     "service_desk": {
      "sip": {
       "uri": "sip:user\\@domain.com",
       "transfer_method": "hangup",
       "transfer_target_header": "Transfer-Target"
      }
```

```
        }
       }
      },
      "agent_available": {
        "message": "Please hold on while I connect you with a live agent."
      },
      "agent_unavailable": {
        "message": "Sorry, I could not find an agent."
      },
      "message_to_human_agent": "The caller needs help resetting their password"
    }
  ]
}
```

Transferring upon failure

To configure transfer on failures, go to the **Advanced** tab in the phone integration settings. The following selections can be configured:

- **Transfer failure message**
- **Disconnect call on transfer failure**

For more information, see Handling call and transfer failures.

Passing watsonx Assistant metadata in SIP signaling

To support loading the conversational history between the caller and watsonx Assistant, the phone integration specifies a value for the `User-to-User` header as a key that can be used with the web chat integration. If `User-to-User` is specified in the `transfer_headers` list, the session history key is sent in the `X-Watson-Assistant-Session-History-Key` header.

The value of the SIP header is limited to 1024 bytes.

How this data is presented in the SIP `REFER` message also depends on the value of `transfer_headers_send_method`.

The example shows the data included as headers:

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 23 REFER
Max-Forwards: 7
Refer-To: sip:user@domain.com
X-Watson-Assistant-Token: 8f817472-8c57-4117-850d-fdf4fd23ba7
User-to-User: 637573746f6d2d757365722d746f2d75736572;encoding=hex
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

If a custom `User-to-User` header is specified, then the session history key is set in the `X-Watson-Assistant-Session-History-Key` header:

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 93809823 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com
User-to-User: 637573746f6d2d757365722d746f2d75736572;encoding=hex
X-Watson-Assistant-Session-History-Key: dev::latest::212033::0a64c30d-c558-4055-85ad-ef75ad6cc29d::978f1fd7-4e24-47d8-adb0-24a8a6eff69e::b5ffd6c2-902f-4658-b586-e3fc170a6cf3::7ad616a350cc48078f17e3ee3df551de
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

This example shows the metadata passed to the `Refer-To` header as query parameters (as defined by SIP RFC 3261).

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
```

```
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 23 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com?User-to-User=637573746f6d2d757365722d746f2d75736572%3Bencoding%3Dhex
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

If a custom `User-to-User` header is specified, then the session history key is set in the `X-Watson-Assistant-Session-History-Key` header.

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293940223
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 93809823 REFER
Max-Forwards: 70
Refer-To: sip:user@domain.com?User-to-User=637573746f6d2d757365722d746f2d75736572%3Bencoding%3Dhex&X-Watson-Assistant-Session-History-Key=dev::latest::893499::dff9c274-adc4-4f63-93de-781166760bf8::978f1fd7-4e24-47d8-adb0-24a8a6eff69e::b5ffd6c2-902f-4658-b586-e3fc170a6cf3::7ad616a350cc48078f17e3ee3df551de
Contact: sip:a@atlanta.example.com
Content-Length: 0
```

Note: For Twilio Flex, the `User-to-User` header uses encoding=ascii.

Playing hold music or a voice recording

To play hold music or to play a recorded message, use the `audio` response type. For more information about response types, see Defining responses with the JSON editor.

You cannot play hold music during a call transfer. However, you might want to play hold music if your assistant needs time to perform processing of some kind, such as calling a client-side action or making a call to a webhook.

The phone integration supports the following properties for the `audio` response type:

Property	Description
`source`	The URL of a publicly accessible `.wav` audio file. The audio file must be single channel (mono) and PCM-encoded, and must have an 8,000 Hz sampling rate with 16 bits per sample.
`channel_options.voice_telephony.loop`	Whether to repeatedly restart the audio playback after it finishes. The default value is `false`.

If you set `channel_options.voice_telephony.loop` to `true`, add a user-defined response with the `vgwActForceNoInputTurn` command. This command instructs the phone integration to initiate a turn with a `vgwNoInputTurn` text without waiting for an input from the caller. In the `vgwNoInputTurn` turn you can initiate a transaction while the caller is on hold. When the `vgwNoInputTurn` turn completes, the looped audio stops.

The following example shows an `audio` response with `loop`=`true`, and a `user_defined` response with the `vgwActForceNoInputTurn` command.

```
{
  "generic": [
    {
      "response_type": "user_defined",
      "user_defined": {
        "vgwAction": {
          "command": "vgwActForceNoInputTurn"
        }
      }
    },
    {
      "response_type": "audio",
      "source": "https://raw.githubusercontent.com/WASdev/sample.voice.gateway/master/audio/musicOnHoldSample.wav",
      "channel_options": {
        "voice_telephony": {
          "loop": true
```

```
      }
     }
    }
   ]
}
```

Enabling keypad entry

If you want customers to be able to send information by typing it on their phone keypad instead of speaking, you can add support for phone keypad entry. The best way to implement this type of support is to enable dual-tone multifrequency (DTMF) signaling. DTMF is a protocol to transmit tones that are generated when a user presses keys on a push-button phone. The tones have a specific frequency and duration that can be interpreted by the phone network.

To start listening for tones as the user presses phone keys, use the `dtmf` response type in an action step. This response type can be added by using the JSON editor.

```
{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {
        "type": "<command type>",
        "parameters": {
          "parameter name": "parameter value",
          "parameter name": "parameter value"
        }
      },
      "channels": [
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}
```

The `command_info` property specifies a DTMF command for the phone integration. The supported commands and their related parameters are as follows.

command_info.type : collect

Instructs the phone integration to collect dual-tone multi-frequency signaling (DTMF) input from a user. This command supports the following parameters:

parameter name	description	required	default
`termination_key`	The DTMF termination key, which signals the end of DTMF input (for example, `#`).	no	n/a
`count`	The number of DTMF digits to collect, which must be a positive integer no larger than 100.	Required if `termintation_key`, or `minimum_count` and `maximum_count`, are not defined	n/a
`minimum_count`	The minimum number of DTMF digits to collect. This property is used along with `maximum_count` to define a range for the number of digits to collect. This value must be a positive integer with a minimum value of 1 and a maximum value less than `maximum_count`.	Required if `terminatation_key` and `count` are not defined.	n/a
`maximum_count`	The maximum number of DTMF digits to collect. This property is used along with `minimum_count` to define a range for the number of digits to collect. When this number of digits is collected, a conversation turn is initiated. This value must be a positive integer no greater than 100.	Required if `termintation_key` and `count` are not defined.	n/a

inter_digit_timeout_count	The amount of time (in milliseconds) to wait for a new DTMF digit after a DTMF digit is received. During an active DTMF collection, this timeout activates when the first DTMF collection is received. When the inter-digit timeout is active, it deactivates the post-response timeout timer. If the inter_digit_timeout_count parameter is not specified, the post-response timer resets after each DTMF digit, and it stays active until either the post-response timeout count is met or the collection completes. This value is a positive integer no higher than 100,000 (or 100 seconds).	no	n/a
ignore_speech	Whether to disable speech recognition during collection of DTMF digits, until either the collection completes or a timeout occurs. If this parameter is true, speech recognition is disabled automatically when the first DTMF signal is received.	no	false
stop_after_collection	Whether to stop DTMF input when the DTMF collection completes. After this command, all DTMF input is ignored until it is reenabled by using the start response type.	no	false

command_info.type : disable_barge_in

Disables DTMF barge-in so that playback from the phone integration is not interrupted when callers press keys. If `disable_barge_in` is enabled, then the keys that are pressed during playback are ignored.

This command has no parameters.

command_info.type : enable_barge_in

Enables DTMF barge-in so that callers can interrupt playback from the phone integration by pressing a key.

This command has no parameters.

command_info.type : send

Sends DTMF signals with the phone integration.

This command supports the following parameters:

parameter	description	required	default
digits	An array of JSON objects where each element represents a DTMF tone to be sent to a caller.	yes	n/a
digits[].code	The event code to send. In addition to the digits 0 through 9, you can specify the following codes: • 10: `*` • 11: `#` • 12: `A` • 13: `B` • 14: `C` • 15: `D`	yes	n/a
digits[].duration	The duration (in milliseconds) of the event.	no	200
digits[].volume	The power level of the tone, in dBm0. The supported range is 0 to -63 dBm0.	no	0
send_interval	An interval (in milliseconds) to wait before the next DTMF tone in the list is sent.	no	200

Examples

This example shows the `dtmf` response type with the `collect` command, used to collect DTMF input.

```
{
  "generic": [
   {
     "response_type": "dtmf",
     "command_info": {
```

```
      "type": "collect",
      "parameters": {
        "termination_key": "#",
        "count": 16,
        "ignore_speech": true
      }
    },
    "channels": [
      {
        "channel": "voice_telephony"
      }
    ]
  }
 ]
}
```

This example shows the `dtmf` response type with the `send` command, used to send DTMF signals.

```
{
 "generic": [
  {
   "response_type": "dtmf",
   "command_info": {
    "type": "send",
    "parameters": {
     "digits": [
      {
       "code": "9",
       "volume": -8
      },
      {
       "code": "11"
      }
     ],
     "send_interval": 100
    }
   },
   "channels": [
    {
     "channel": "voice_telephony"
    }
   ]
  }
 ]
}
```

Transferring the conversation to the web chat integration

You can transfer the caller from the current phone call to a web chat session by using the `channel_transfer` response type.

The assistant sends an SMS message to the caller that includes a URL that the caller can tap to load the web chat widget in the phone's browser. The web chat session displays the history of the phone call and can start the process of collecting information that is needed to complete the transaction.

This is useful in situations when the customer can provide information more easily in writing than by speaking (for example, changing an address).

After the transfer successfully completes, the caller can hang up the phone and continue the conversation by using web chat.

The `channel_transfer` response type can be used with the phone integration only if the *SMS with Twilio* integration is also configured for the assistant.

```
{
 "generic": [
  {
   "response_type": "text",
   "values": [
    {
     "text": "I will send you a text message now with a link to our website."
    }
   ],
   "selection_policy": "sequential"
  },
```

```
    {
      "response_type": "channel_transfer",
      "message_to_user": "Click the link to connect with an agent using our website.",
      "transfer_info": {
        "target": {
          "chat": {
            "url": "https://example.com/webchat"
          }
        }
      }
    }
  ]
}
```

Ending the call

You can instruct your assistant to end a phone call by using the `end_session` response type, as shown in this example.

```
{
  "generic": [
    {
      "response_type": "end_session"
    }
  ]
}
```

You can optionally include custom headers to include with the SIP `BYE` request that is generated when the phone integration receives this response type.

This example shows the `end_session` response type with custom SIP headers:

```
{
  "generic": [
    {
      "response_type": "end_session",
      "channel_options": {
        "voice_telephony": {
          "sip": {
            "headers": [
              {
                "name": "Customer-Header1",
                "value": "Some-Custom-Info"
              },
              {
                "name": "User-to-User",
                "value": "XXXXXX"
              }
            ]
          }
        }
      }
    }
  ]
}
```

Sending a text message during a phone conversation

In situations when it is easier to communicate accurately in writing than by transcribing voice output, you can send a text message during an ongoing voice. For example, an address or directions.

Note: Before you can send SMS messages during a phone call, you must set up the *SMS* integration. For more information, see *Integrating with SMS*.

When you exchange a text with a customer during a conversation, the assistant initiates the SMS message exchange. A text message is sent to the user and asks for the user to respond.

To send a specific message from an action step, use the `user_defined` response type with the `vgwActSendSMS` command:

```
{
  "generic": [
    {
      "response_type": "text",
      "values": [
        {
          "text": "I will send you a text message now."
        }
      ],
      "selection_policy": "sequential"
    },
    {
      "response_type": "user_defined",
      "user_defined": {
        "vgwAction": {
          "command": "vgwActSendSMS",
          "parameters": {
            "message": "To send me your street address, respond to this text message with your address."
          }
        }
      }
    }
  ]
}
```

You can specify any of the following parameters in the `parameters` object:

Parameter	Type	Description
message	string	The text of the SMS message to send. Required.
mediaURL	list	A list of URLs for media files to be sent with the message as MMS attachments. Optional.
tenantPhoneNumber	string	The phone number that is associated with the tenant. The format of the number must match the format that is required by the SMS provider. If no `tenantPhoneNumber` value is provided, the tenant ID from the phone integration configuration for the active call is used. Optional.
userPhoneNumber	string	The phone number to send the SMS message to. The format of the number must match the format that is required by the SMS provider. If no `userPhoneNumber` value is provided, the voice caller's phone number from `From` header of the incoming SIP `INVITE` request is used. Optional.
setAsInputText	boolean	Whether to send a sms message from the user in `input.text`. If you specify `true`, the sms message from the user will be sent in `input.text`. Otherwise, `input.text` will be set to `vgwSMSMessage` and the SMS message will be sent as an integration variable and a context variable. Optional. Default: false.

If your *SMS* integration supports more than one SMS phone number, or you are using a SIP trunk different from your SMS provider, be sure to specify the phone number that you want to use to send the text message. Otherwise, the text is sent by using the same phone number that was called.

After the assistant receives an SMS message, a new conversation turn is initiated with the text input `vgwSMSMessage`. This input indicates that a message was received from the caller. The text of the customer's message is included as the value of the `vgwSMSMessage` context variable and the `sms_message` integration variable.

If the assistant is unable to send an SMS message to the caller, a new turn is initiated with the text input `vgwSMSFailed`. This input indicates that an SMS message could not be sent to the caller. You can design your assistant to handle such a failure by creating actions that are triggered by the input text `vgwSMSFailed`.

```
$ {
  "input": {
    "text": "vgwSMSMessage",
    "integrations": {
      "voice_telephony": {
        "sms_message": "230 Leigh Farm rd"
      }
    }
  },
  "context": {
```

```
    "skills": {
      "main skill": {
        "user_defined": {
          "vgwSMSMessage": "1545 Lexington Ave."
        }
      }
    }
  }
}
```

Here's an example of a turn request when `setAsInputText` is set to `true`:

```
$ {
  "input": {
    "text": "230 Leigh Farm rd",
    "integrations": {
      "voice_telephony": {
        "sms_message": "230 Leigh Farm rd"
      }
    }
  },
  "context": {
    "skills": {
      "main skill": {
        "user_defined": {
          "vgwSMSMessage": "1545 Lexington Ave."
        }
      }
    }
  }
}
```

Defining a sequence of phone commands

If you want to run more than one command in succession, include multiple responses in the `generic` array. These commands are processed in the order in which they are specified in the array.

This example shows two responses: first, a text response; and second, an `end_session` response to end the call.

```
{
  "generic": [
    {
      "response_type": "text",
      "values": [
        {
          "text": "Goodbye."
        }
      ],
      "selection_policy": "sequential"
    },
    {
      "response_type": "end_session"
    }
  ]
}
```

Injecting custom values into CDR log events

If you are using a log webhook to log call detail record (CDR) events, you can use the `cdr_custom_data` context variable to add custom data to logged events. You can use this method to record data during a call (for example, to indicate the completion of a specific task).

To log custom CDR data, use the JSON editor to edit the context. Define `cdr_custom_data` as a child of the `context.integrations.voice_telephony` object, as in this example:

```
"context": {
  "integrations": {
    "voice_telephony": {
      "cdr_custom_data": {
```

```
        "key1": "value1",
        "key2": "value2"
      }
    }
  }
}
```

The `cdr_custom_data` object can contain any valid JSON data.

When you generate a CDR report, the custom data is included in the `injected_custom_data` field, as in this example:

```
{
  "payload": {
  ...
    "injected_custom_data": {
      "key1": "value1",
      "key2": "value2"
    }
  ...
  }
}
```

For more information about the structure of the CDR log event payload, see CDR log event reference.

Merging and deleting custom CDR data

Each time the `cdr_custom_data` object is defined by an action, the new data is merged with any previously existing data. New values that are specified for previously defined properties overwrite the previous values, and any new properties are added; otherwise, the previously defined data is unchanged.

To remove a previously defined property, you must explicitly set it to an empty value, as in this example:

```
"context": {
  "integrations": {
    "voice_telephony": {
      "cdr_custom_data": {
        "key1": ""
      }
    }
  }
}
```

Access phone integration context variables from your action

If you want to access the phone integration context variables, use the JSON editor to edit the context.

The following example shows how to access the user phone number (the phone number that the call was received from):

```
"context": {
  "variables": [
    {
      "value": {
        "expression": "${system_integrations.voice_telephony.private.user_phone_number}.replace('+','')"
      },
      "skill_variable": "user_phone_number"
    }
  ]
}
```

For more information, see Phone integration context variables.

Troubleshooting and Logs

Troubleshooting

Find solutions to problems that you might encounter when using the phone integration.

- A *Forbidden* message: The phone number that you specified when you configured the phone integration cannot be verified. Ensure that the number fully matches the SIP trunk phone number.
- A lot of latency between caller questions and Watson answers: Most likely coming from latency that is caused by one of the Watson services, you can view logs in IBM Log Analysis. At the end of each call, you see a `CWSGW0160I: Call was ended.` event. Expand on this entry to see a summary of the `max_response_milliseconds` and details of the `assistant_interaction_summaries`, which helps you identify the service the latency is coming from.

  If your plan allows, you can also look at the Call Detail Record (CDR) to determine which service is misbehaving. For more information, see Call Detail Records (CDRs).

Viewing logs

Log events from the components used by the phone integration are written to IBM Cloud Logs. To view these logs, create an instance and configure platform logs to monitor the region where your service instance is hosted.

For instructions on setting up an instance, see Provisioning an instance.

> **Note:** Currently, only the Phone and SMS integrations write logs to the IBM Cloud Logs dashboard.

Once the instance is created, follow these steps to access log information:

1. Go to the IBM Cloud Logging page.
2. Select the Logging instance to use.
3. From the panel, choose **Getting Started**.
4. Click **Add Data Sources** > **Configure Platform Logs**.
5. Set the target region for log viewing, select the IBM Cloud Logs instance, and click **Save**.
6. To open the IBM Cloud Logs console, return to the IBM Cloud Logging page, and click **Dashboard**.
7. From the menu, select **Explore Logs**.

You can apply filters or search the logs by values such as phone number or instance ID.

Call Detail Records (CDRs)

The phone integration can generate call detail record (CDR) events, which contain summary information about a single call. Call detail records are configured through a webhook. For more information, see Logging activity with a webhook.

For more information about the structure of the CDR event payload, see CDR log event reference.

You can also inject custom data into the CDR event. For more information, see Injecting custom values into CDR log events.

Deploying to other channels

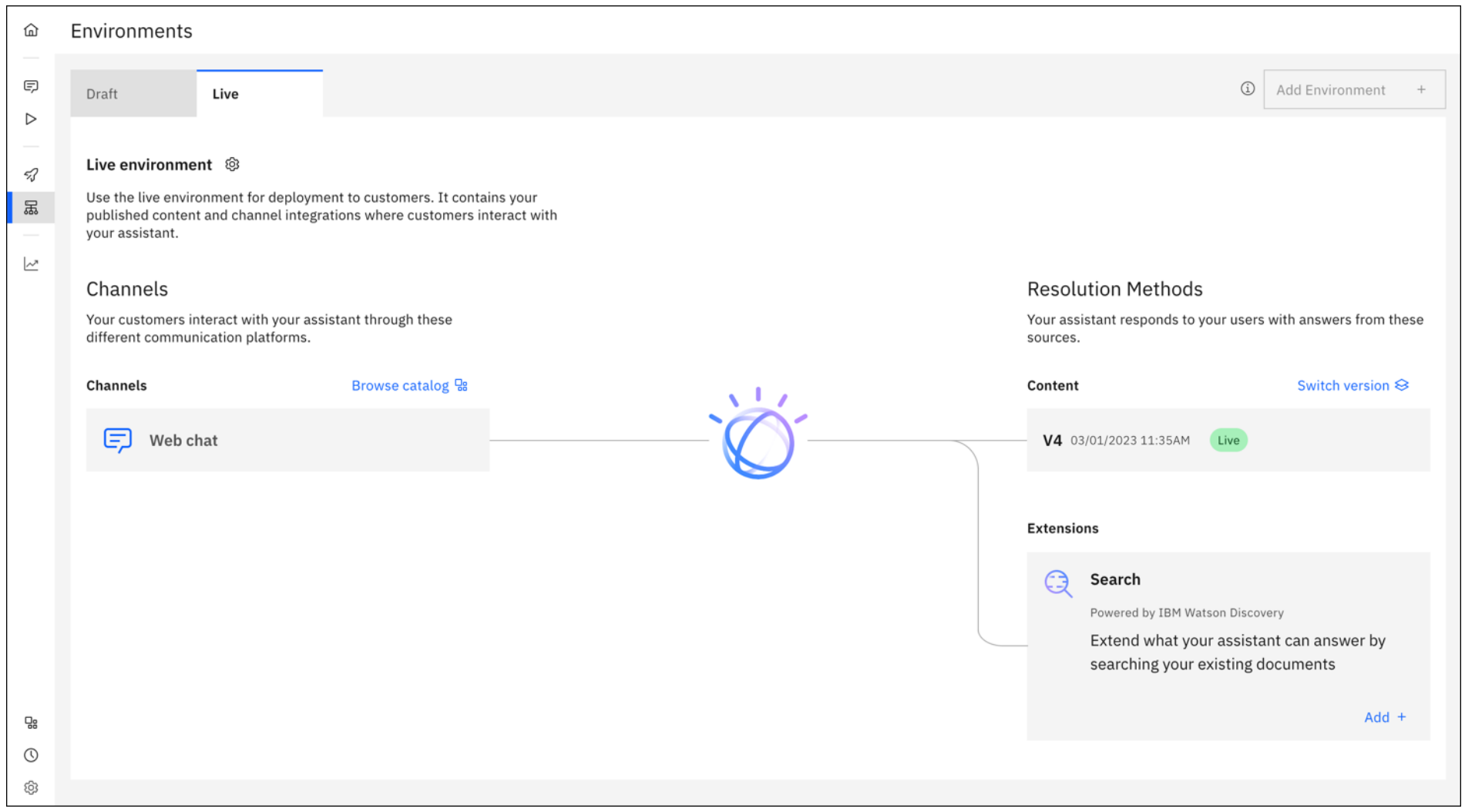
Deploying your assistant

IBM Cloud

Your assistant routes your customer's questions and requests to the correct resolution source. However, before your assistant can properly route requests, you must complete the following steps:

1. Write content for your assistant
2. Publish the content
3. Connect an integration to your assistant
4. Deploy the assistant to a channel

After you connect an integration and deploy to a channel, you make your assistant available to your customers. You can find a visual overview of your progress toward completing these steps on the **Live environment** page:



Live environment

Reviewing your connected channels

Channels represent the locations or communication platforms where your assistant interacts with your users. Common examples of channels include the phone, a website, or Slack. If you do not connect your assistant to a channel, your users are not able to access the assistant.

You can review your connected channels in your environment, for example:

- **Draft environment**: Channels that are connected to this environment are available only to your internal team for testing and not to your customers.
- **Live environment**: Channels that are connected to this environment represent the public-facing experience of your assistant and are available to your customers.
- **Multiple environments**: If you are using multiple environments, channels are available only to your internal team for testing and not to your customers.

For more information, see Environments and Adding and using multiple environments.

You can test integrations from the draft environment and interact with your draft web chat on the **Preview** page. When you first create a new assistant, the assistant automatically connects to the web chat channel in your live environment. However, the assistant itself is not available to your customers until you embed the web chat JavaScript in the header of your website. For more information, see Adding the web chat to your website.

Connecting your assistant to a new channel

All of the channels you can connect your assistant to are available in the **Integrations** catalog. Two types of integrations are available from the **Integrations** catalog:

- Channels: The location where your assistant interacts with your users, for example, over the phone, on a website, or in Slack. At least one channel is required for every assistant.
- Extensions: Add-ons to the end experience that help solve specific user problems, for example, connecting to a human agent or searching existing help content. Extensions are not required for an assistant, but they are recommended.

When you add a channel to your assistant, at least two instances of the channel are created. One instance of the channel is connected to the draft environment and the other instance is connected to the live environment. If you are using multiple environments, instances of the channel are added to your extra environments. To connect your assistant to a new channel, go to the **Integrations** catalog. For more information about adding integrations to your assistant, see Adding integrations.

Although a channel always exists in both environments, you can configure your integration separately in each environment. For example, you can test integrations on your draft environment before you go live with any integration configuration. After you add an integration, you must set it up to use it with your assistant. The **Finish setup** icon appears on any integration that you added but didn't yet set up.

You have multiple options for deploying your assistant, depending on how you want your customers to interact with it. In most cases, an assistant is deployed by using one of the following integrations:

- Web chat integration: The web chat integration provides a secure and highly customizable widget that you can add to your website. You can configure how and where the web chat widget appears, and you can use theming to align it with your branding and website design. If a customer needs help from a person, the web chat integration can transfer the conversation to an agent.
- Phone integration: The phone integration enables your assistant to converse with customers on the phone by using the IBM Watson Text to Speech and Speech to Text services. If your customer asks to speak to a person, the phone integration can transfer the call to an agent.

Updating and managing channels

Each channel has specific settings that you can adjust to adapt the end experience for your user. You can edit these settings by selecting the channel in an environment, or in **Integrations**.

If you make an update to a channel in the draft environment, the same channel in the live environment is not affected in the live environment. Similarly, if you make an update to a channel in the live environment, the same channel in the draft environment is not affected. If you select a channel from the **Integrations** page, you are asked to select which environment you are editing.

For more information about editing your web chat integration, see Web chat overview.

Deleting channels

To delete a channel:

1. Open the **Integrations** page
2. Click the options menu icon ⋮ on the integration.
3. Choose **Delete from assistant**. You then need to confirm the deletion.
4. In the confirmation, type `DELETE`, then click the **Delete** button.

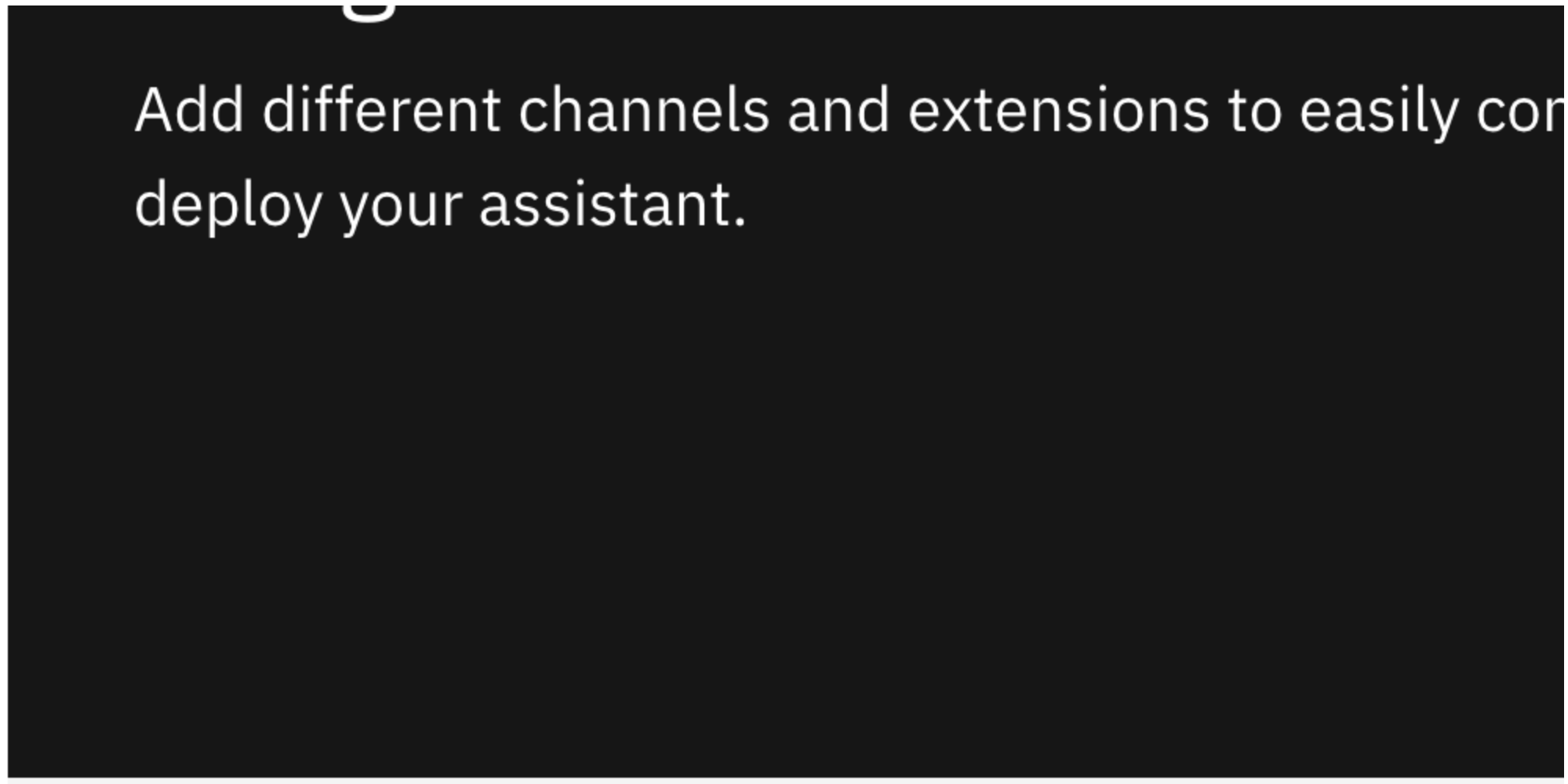
If you deployed your assistant to a channel, then deleting the channel does not remove the assistant from the channel. For example, if you deploy web chat, you paste the JavaScript snippet into the HTML header of your website. Deleting your channel disconnects your content from the customer experience that is shown on your website. Be sure to remove the JavaScript snippet before you delete the channel.

Deleting a channel is final and deletes the channel from all environments. Think twice before you delete an integration because each new integration that you might add starts with default settings. If you delete your channel and then decide you want it back in a previous state, you must rebuild the channel after you add it back.

Adding integrations

Add integrations to your assistant so that you can publish your bot to the channels where your customers go for help.

To deploy an assistant to customers, a channel integration must be added. By default, a web chat integration is created, allowing an assistant to be embedded in a website. Other channel integrations are available in the **Integrations** catalog. For more information, see Adding the web chat to your website.



Essential channels

Add our most utilized methods of deploying assistants.

These channels support additional customization and advanced integrations.

When you add an integration, that integration is added to both the draft and live environments, or to all your environments if you are using multiple environments. Test content and integrations before you deploy your assistant to customers. For more information about adding integrations to your assistant, see Adding integrations. After a live channel is added and configured, it is ready to deploy your assistant on its corresponding platform.

Add an integration

Follow these steps to add integrations to your assistant:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Scroll to see available integrations.

   Why do I have the *web chat* **integration?** This integration is provisioned and added automatically to your first assistant only.

   watsonx Assistant comes with the web chat integration, which is an engaging and fully extensible front-end client that can be added to your website in minutes. It is designed to handle advanced conversational scenarios, including rich responses, such as images, video, and iframes, suggestions when the user gets stuck, and live agent escalation.
3. For any integration that you want to add, click **Add**. The options include:

   - Web chat
   - Phone
   - Facebook Messenger
   - Genesys Bot Connector
   - Microsoft Teams
   - Slack
   - SMS
   - WhatsApp with Twilio
   - Search

   You can also set up live agent integrations by going to the **Live agent** tab from the **web chat** tile. The options include:

   Web chat live agent integrations

   - Web chat with Genesys Web Messenger
   - Web chat with NICE CXone Digital First Omnichannel Live Chat
   - Web chat with Salesforce support
   - Web chat with Zendesk support

   Web chat live agent reference implementations

   - Genesys Cloud
   - Kustomer
   - NICE inContact
   - Oracle Cloud B2C
   - Twilio Flex
   - Bring your own (starter kit)

   Phone live agent integrations

   - Phone with Genesys Cloud
   - Phone with NICE CXone
   - Phone with Twilio Flex
4. Follow the instructions provided on screen to complete the integration process.

Note: Plus For environments where private endpoints are in use, keep in mind that these integrations send traffic over the internet.

How live agent integrations work

Watch a 4-minute video about integrating your assistant with a live agent integration, such as Zendesk:

View video: Zendesk Integration: watsonx Assistant

To learn how live agent integrations with your assistant can benefit your business, read Customer Service Just Got Smarter.

Integrating with Facebook Messenger

IBM Cloud

Facebook Messenger is a messaging application that helps businesses and customers communicate directly with one another.

After you create an action, you can integrate your assistant with Facebook Messenger.

> ⚠ **Important:** Currently, no method exists to identify or delete data that is associated with a specific user who interacts with the assistant through Facebook Messenger. Do not use this integration method for deployments that must be compliant with the General Data Protection Regulation (GDPR).

Before you begin

To integrate your assistant with Facebook, you must have the `Admin` role on Facebook's Meta App Development. For more information, see Meta App Development.

To integrate your assistant with Facebook, do the following steps:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the **Facebook Messenger** tile.
3. Click **Confirm**.
4. Follow the instructions that are provided on the screen to complete the integration process.

> **Note:** When a field that is required for authentication is changed, all entries in the related fields must be filled and validated again.

Action considerations

The rich responses that you add to the action are displayed in a Facebook app as expected, with the following exceptions:

- **Connect to live agent**: This response type is ignored.
- **Image**: This response type embeds an image in the response. A title and description are not displayed before the image, whether or not you specify them.
- **Option**: This response type shows a list of options that the user can choose from.
  - A description is not displayed, whether you specify one or not.
  - After a user clicks one of the buttons, the button choices disappear and are replaced by the user input that is generated by the user's choice. If the assistant or the user enters new input, then the button-generated input disappears. Therefore, if you include multiple response types in a single response, position the option response type last. Otherwise, content from subsequent responses, such as text from a text response type, will replace the button-generated text.
  - The title is automatically taken from the text of the relevant step of the action where options are listed.

Chatting with the assistant

To start a chat with the assistant, complete the following steps:

1. Open Facebook Messenger.
2. Type the name of the page you created earlier.
3. After the page comes up, click it, and then start chatting with the assistant.

The welcome action is not processed by the Facebook Messenger integration. The welcome message is not displayed in Facebook Messenger like it is in the assistant preview. It is not triggered from here because nodes with the `welcome` special condition are skipped in action flows that are started by users. Facebook Messenger waits for the user to initiate the conversation.

The action flow for the current session is restarted after 60 minutes of inactivity (5 minutes for Lite and Standard plans). This means that if a user stops interacting with the assistant, after 60 (or 5) minutes, any context variable values that were set during the previous conversation are set to null or back to their default values.

Note: Only the page administrator can interact with the Facebook Messenger chatbot until Facebook approves it. After Facebook approves it, any page visitor can interact with it.

Integrating with Genesys Audio Connector

IBM Cloud

You can integrate the Genesys Audio Connector with your assistant to stream the conversation audio between the assistant and Genesys Cloud.

Before you begin

You must have the following prerequisites before you start integrating your assistant with Genesys Audio Connector:

- A new account or log in to an existing Genesys Cloud account with access to Genesys Architect and the correct region at the Genesys Cloud portal.
- The `Admin` role in the Genesys Cloud organization.

For more information on Genesys Cloud roles and permissions see, Roles and permissions overview.

Create the Audio Connector integration in your assistant

1. Go to the **Integrations** page by clicking the integrations icon in the left menu.

Important: Store the credentials because you cannot see them after you click **Save**. You require these credentials to set up the Genesys Audio Connector.

1. Click **Add** on the **Phone** tile.
2. Click **Genesys Audio Connector** tile.
3. In the **Credentials** section, copy and store the automatically generated credentials in the following fields:
    - **API key**
    - **Client secret**
    - **Genesys audio connect URI**
    - **Bot Connector ID**
4. Click **Save and Exit**.

Set up Audio Connector to integrate assistant

To set up Genesys Audio Connector, complete the steps in the Configure and activate Audio Connector in Genesys Cloud topic in the Genesys Cloud documentation.

Important: In the Genesys **Admin** page, go to **Integrations** > **Configuration** to add the **Genesys audio connect URI** value that you copied when you created the Audio Connector integration in your assistant.

Important: In the Genesys **Admin** page, go to **Integrations** > **Credentials** to add the credentials for the **API key** and **Client secret** fields that you copied when you created the Audio Connector integration in your assistant.

Call flow

Use the **Call Audio Connector** action in Genesys to activate the Audio Connector integration in your assistant.

For more information, see Call Audio Connector action.

After you get the Genesys audio connect URI from the **Credentials** section, do the following:

1. Go to the Genesys Admin page.
2. Under Architect, click **Architect**, and create an **Inbound Call Flow**.
3. In the Toolbox, click **Bot**, and then **Call Audio Connector**.
4. Enter a name for your Call Flow.
5. Choose your Audio Connector integration from the **Integration** drop-down menu.
6. You must copy the stored **Bot Connector ID** field of the **Credentials** section and paste it into the **Connector ID** field. `<instance-id>/connect?version=<api-version>`
7. Enter session variables that you want to be passed to and from watsonx Assistant. For more information, see Context Sharing through session variables.
8. Click at the bottom of the Flow Diagram to create a Terminating Action. For example, `Disconnect`.
9. Click **Save** and **Publish**.

Call routing

Create a call routing to direct incoming calls to your Genesys Call Flow.

1. Go to the Genesys Admin page.
2. Go to **Call Routing** to create a call routing.
3. Enter a name for your call routing.
4. Choose **Division**.
5. Choose the call flow that you configured in the previous step from the **Route to** drop-down menu.
6. In the call flow, assign the telephone number to which you want to route the call.
7. Click **Create**.

Context-sharing through session variables

From the Audio Connector node in the Genesys Architect flow, you can specify session variables that can be used to pass information to watsonx Assistant. You can specify both Input and Output parameters. For the integration, both of these parameters are merged into a single object under the `context` object.

Both Input and Output parameters are available in the watsonx Assistant `context`, and the information is shared on each turn. For example, the `context` made available in watsonx Assistant is:

```
$ {
  "context": {
    "integrations": {
     "genesys_audio_connector": {
      "user_id": "<SENT FROM GENESYS>",
      "some_variable": "<SET_FROM_WATSON_ASSISTANT>"
     }
    }
  }
}
```

Access the input parameters in your watsonx Assistant with the ${system_integrations.genesys_audio_connector.user_id} session variable.

To set output parameters, use an expression such as ${system_integrations.genesys_audio_connector.some_variable}, which you can access later in your flow.

Do the following steps to set an output variable:

1. Go to the action step where you want to define the variable.
2. Click **Set new variable** and select **Expression**.
3. Enter the following expression: `${system_integrations.genesys_audio_connector.<variable_name>} = "<variable_value>"`

   Example:

```
$ ${system_integrations.genesys_audio_connector.some_variable} = "this is an output variable"
```

In this example, the variable name is `some_variable`, with the value set to `this is an output variable`.

Ending the Genesys Audio Connector flow

After receiving the audio conversation from the user, the Audio Connector node in Genesys Architect facilitates exchange of messages between the user and assistant until the end of conversation. To send the audio conversation back to Genesys, you must use the `end_session` response type.

```
$ {
  "generic": [
    {
      "response_type": "text",
      "values": [
        {
          "text": "You have ended the call."
        }
      ]
    },
    {
      "response_type": "end_session"
    }
  ]
}
```

Response types

After integrating your watsonx Assistant with Genesys Audio Connector, you can use the following response types in the assistant:

- text
- option
- end_session
- speech_to_text
- text_to_speech
- start_activities
- stop_activities
- dtmf
- user_defined
- connect_to_agent
- image
- audio

Integrating with Genesys Bot Connector

IBM Cloud

Use the Genesys Bot Connector integration to connect your assistant with Genesys.

The Genesys Bot Connector enables Genesys Architect Flow designers to integrate their messaging and conversation flows with any third-party virtual assistant.

Before you begin

You must comply with the following requirements before you start integrating your assistant with Genesys Bot Connector:

- A new account or log in to an existing Genesys Cloud account with access to Genesys Architect and the correct region at the Genesys Cloud portal.
- The `Admin` role in the Genesys Cloud organization. For more information on Genesys Cloud roles and permissions see, Roles and permissions overview.

Set up Bot Connector in Genesys

1. Go to the Genesys Admin page.

2. Under **Integrations**, click **Integrations**.
3. On the Integrations page, click **+Integrations**.
4. Search for Genesys Bot Connector, and click **Install**.
5. In the Details tab, enter a name and any notes for your Bot Connector.
6. Click **Copy the Integration ID**, and save. You need the **Integration ID** to create the watsonx Assistant Genesys Bot Connector integration.
7. In the Configuration tab, under Properties, enter a placeholder **Bot Connector Handle Utterance URI**. You need to come back and update this URI after a watsonx Assistant Genesys Bot Connector integration is created.
8. In the Credentials tab, use **Configure** to create a credential field with **Field Name** as **x-watson-genesys-verification-token** and with **Value** as the secret you set in watsonx Assistant. You need this **value** to create the watsonx Assistant Genesys Bot Connector integration. Copy and keep the **token** and **value**, so you can paste them into the **Verification Token** field when you integrate with watsonx Assistant.

   > **Note:** You cannot see these credentials again after clicking Save.

9. Click OK and Save.

Keep the Genesys web page open in a web browser tab so you can refer to and complete fields as setup progresses.

Integrate watsonx Assistant with Genesys

Create the Bot Connector integration

1. Go to the watsonx Assistant **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the **Genesys Bot Connector** tile.
3. Click **Confirm**.

Connect watsonx Assistant to Genesys

1. From the Genesys site, you need your Genesys OAuth credentials.

   OAuth credentials are on the Genesys Admin page under Integrations > OAuth. You need credentials with Grant Type **Client Credentials** and a role that has BotConnector permissions.

   Copy and keep the following values, so you can paste them into the *Genesys Bot Connector* integration setup page.

   - **Client ID**
   - **Client Secret**

2. Return to the watsonx Assistant *Genesys Bot Connector* integration setup page, and then click **Next**.
3. Enter the required fields, and then click **Next**.
   - **Client ID**
   - **Client Secret**
   - **Verification Token** (token and value from Genesys Bot Connector setup)
   - **Integration ID**
   - **API URI** (the Genesys API server for your region, for example: `https://api.regionxyz.mypurecloud.com`)

> **Note:** For security reasons, the authentication fields are removed from view after initial setup.

Configure your Genesys Bot Connector

1. Copy the value generated in the **Webhook URI** field.
2. Go to the Genesys Bot Connector Configuration tab you left open. Under Properties, replace the placeholder *Bot Connector Handle Utterance URI* you entered previously with this **Webhook URI** value.
3. Click **Finish**.

> **Note:** If a field required for authentication is changed, then all entries in related fields must be filled and validated again.

Chat with the assistant

To start a chat with the assistant, complete the following steps:

1. Open Genesys Architect, and create an **Inbound Message Flow**.
2. In the Toolbox, click **Bot**, and then **Call Bot Connector**.
3. Select the values:
    - Bot Integration:
    - Bot Name: IBM watsonx Assistant Bot Connector
    - Bot Version: 1.0
4. Enter session variables you want to be passed to and from watsonx Assistant. For more information, see Context Sharing through Session Variables.
5. Output of the Bot Connector is `Success` or `Failure`. You should branch your Genesys flow upon exit from the Bot Connector according to the output **Intent of the Bot Connector**. For more information, see Conditioning and Output Branching.

Context-sharing through session variables

From the Bot Connector node in the Genesys Architect flow, you can specify session variables that can be used to pass information to watsonx Assistant. You can specify both Input and Output parameters. For the integration, both of these parameters are merged into a single object under the `context` object.

Both Input and Output parameters are available in the watsonx Assistant `context`, and the information is shared on each turn. For example, the `context` made available in watsonx Assistant is:

```
$ {
  "context": {
    "integrations": {
      "genesys_bot_connector": {
        "user_id": "<SENT FROM GENESYS>",
        "some_variable": "<SET_FROM_WATSON_ASSISTANT>"
      }
    }
  }
}
```

For Input parameters, you can access the session variable in watsonx Assistant as follows:

- **For Dialog**: Use `$integrations.genesys_bot_connector.user_id`.
- **For Actions**: Use `${system_integrations.genesys_bot_connector.user_id}`.

For Output parameters, you can assign the session variables to a state variable in Genesys (for example, `State.some_variable` and access them later on in your flow).

Variables can be read and set from both Dialog and Action skills.

Set slot parameters by the watsonx Assistant Bot Connector

watsonx Assistant Bot Connector sets the following output parameter based on the last conversation turn when the conversation ends.

Parameter	Description
`Intent`	watsonx Assistant is set to the recognized intent by Dialog skill as the slot value for the Success intent.

Conditioning and output branching

From Genesys Architect, you can use `Logical` functions to branch out your flow based on the context shared back from watsonx Assistant. For example, watsonx Assistant returns an `Intent` parameter that is saved to `State.Intent`.

In architect, you can add a `Switch` action to branch out to different scenarios. For example, you can configure a case where `State.Intent == "connect_to_agent"`, and then branch out.

Ending the Genesys Bot Connector flow

When the conversation reaches the Bot Connector node in Architect, Genesys proxies messages between the user and watsonx Assistant, and continues

until the conversation ends. To pass the conversation back to Genesys, you need to use the `connect_to_agent` response type or the `end_session` response type.

```
$ {
  "output": {
    "generic": [
      {
        "response_type": "text",
        "values": [
          {
            "text": "You have ended the call."
          }
        ]
      },
      {
        "response_type": "end_session"
      }
    ]
  }
}
```

```
$ {
  "output": {
    "generic": [
      {
        "response_type": "text",
        "values": [
          {
            "text": "Connecting you to an agent."
          }
        ]
      },
      {
        "response_type": "connect_to_agent"
      }
    ]
  }
}
```

User-defined response type

The `user_defined` response type allows you to pass a custom response back to Genesys. For example, this can be used to pass `Cards` and `Carousels` back to Genesys which do not have a watsonx Assistant equivalent response type. An example of sending back a `Card` back to Genesys is:

```
$ {
  "output": {
    "generic": [
      {
        "type": "user_defined",
        "user_defined": {
          "replyMessages": [
            {
              "type": "Structured",
              "content": [
                {
                  "contentType": "Card",
                  "card": {
                    "title": "Card title",
                    "description": "Card description",
                    "image": "http://www.samplesite.com/photo/1234.jpg",
                    "actions": [
                      {
                        "type": "Link",
                        "text": "Link display text",
                        "url": "http://www.samplesite.com"
                      },
                      {
                        "type": "Postback",
                        "text": "Postback display text",
                        "payload": "Postback text"
```

```
              }
            ]
          }
        }
      ]
    },
    "response_type": "user_defined"
  }
 ]
}
}
```

Response types

These response types are supported and displayed as expected when your assistant is deployed for the Genesys Bot Connector channel.

- connect_to_agent
- end_session
- option
- text
- user_defined

Integrating with Microsoft Teams

IBM Cloud

Add a chatbot to Microsoft Teams to create and customize a productive hub where content, tools, and people come together to chat, meet, and collaborate.

After you create an action, you can use this integration to connect your assistant with Microsoft Teams.

Before you begin

To integrate your assistant with Microsoft Teams, you must have the `Global Administrator`, or `Teams Administrator` roles. For more information, see App permission policies.

Sign up for a Microsoft 365 Developer Administrator email address, if you don't have one:

1. Go to the Microsoft Dev Center.
2. Click **Join now** to become a member of the Microsoft 365 Developer program.
3. On the Dashboard, click **Set up E5 subscription**, and select an **Instant** or a **Configurable** sandbox.
4. Copy the email address listed under **Administrator**. Your admin credentials are required at several points of setup.

Adding the Microsoft Teams integration

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the **Microsoft Teams** tile.
3. Click **Confirm**. The **Get Started** page provides a summary of the setup process.
4. Click **Next** to begin app registration.

App registration

1. Go to the Microsoft Azure portal, and log in with your admin credentials.
2. On the **App registrations** page, click **New registration**.
3. On the **Register an application** page, enter a name, select the multi-tenant option that applies to your app, and then click **Register**.
4. Copy the application ID from the **Overview** page of your app, and paste it into the **App registration** field of your watsonx Assistant Microsoft Teams integration.
5. On the same Microsoft Azure **Overview** page, click the hyperlink **Add a certificate or secret** next to Client Credentials.

6. On the **Certificates & secrets** page for token creation, click **New client secret**. Enter a description and then select **Recommended 180 days**. Click **Add**.
7. Copy the string under **Value** and paste into **Client secret value** on the **App registration** page of your watsonx Assistant Microsoft Teams integration. Note: You *must* generate a new value before the current one expires on day 180.
8. Click **Next** to create your bot.

Create your bot

1. Go to the Microsoft Bot Framework developer portal, and log in with your admin credentials.
2. On the **Tell us about your bot** page, complete your bot profile.
3. Copy the generated endpoint from the **Create your bot** page of your watsonx Assistant Microsoft Teams integration and paste into the **Messaging endpoint** field of the **Configuration** section.
4. Select **Multi-Tenant** as the app type.
5. Copy and paste your app ID, and then click **Register**.
6. On the **Connect to channels** page, click **Configure Microsoft Teams channel** in the **Add a featured channel** section.
7. On the **Configure Microsoft Teams** page, specify options in the **Messaging**, **Calling**, and **Publish** tabs that fit your bot needs, and then click **Save**.
8. In your watsonx Assistant Microsoft Teams integration, click **Next** to create your Teams app.

Create your Teams app

1. Go to the Microsoft Teams Developer Portal, and log in with your admin credentials.
2. On the **Apps** tab, click **New App**.
3. Enter a name, and click **Add**.
4. On the **Basic information** page, enter app names, app ID, descriptions, developer information, and app URLs, and then copy and paste your app ID into **Application (client) ID**. Click **Save**.
5. In the **Configure** section, select **App features**, and then **Bot**.
6. On the **Identify your bot** page, select your bot.
7. In the **Select the scope in which people can use this command** section, select **Personal**, **Team**, and **Group Chat**.
8. Click **Save**, but keep the window open.
9. In your watsonx Assistant Microsoft Teams integration, click **Finish**.
10. Click **Publish** to publish your bot.

Note: When a custom extension is run during a conversation in Microsoft Teams, the client receives pre-execution steps only after the call to the extension is completed. For more information, see calling a custom extension.

Publishing your Teams app

1. In the Microsoft Teams Developer Portal window where you created and saved your Teams app, click **Publish** to publish your bot.
2. Click **Download the app package**.
3. Go to Microsoft Teams, and log in with your admin credentials.
4. Click **Apps** in the sidebar menu, and then click **Manage your apps** and **Upload an app**.
5. Select **Upload a custom app** and specify the app package .zip file you downloaded.
6. Click **Add** to finish.
7. Test your actions and responses in the **Chat** section of your Teams app.

Response types

These response types are supported and displayed as expected when your assistant is deployed for Microsoft Teams direct messages, channels, or group chats.

- date
- image
- option
- suggestion
- text

Integrating with Slack

IBM Cloud

Slack is a cloud-based messaging application that helps people collaborate with one another.

After you create an action, you can integrate your assistant with Slack.

When integrated, depending on the events that you configure the assistant to support, your assistant can respond to questions that are asked in direct messages or in channels where the assistant is directly mentioned.

An example and instructions on how to create a Slackbot using watsonx Assistant, Slack, and Db2 are given in the solution tutorial, Build a database-driven Slackbot.

Before you begin

To integrate Slack with your assistant, you must have a Slack app and the necessary roles and permissions:

Roles	Permissions
Workspace or Org owner	View information Post information Perform actions

To create a slack app, see Quickstart: Start a workflow.

For more information on roles and permissions, see Slack-Getting started.

Adding the Slack integration

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the **Slack** tile.
3. Then, click **Add** again.

Get started

There are four steps to setting up Slack:

- Set up your Slack bot
- Connect watsonx Assistant to Slack
- Configure your Slack bot
- Connect your assistant

Set up your Slack bot

1. Go to the Your Apps page on the Slack website, and then click the app you want to use or create a new one.

Connect watsonx Assistant to Slack

1. On the **Slack app settings** page, go to the **Basic Information** tab and scroll down to the **App Credentials** section.
2. Copy your verification token and paste it in the **Assistant setup** page.
3. On the **Slack app settings** page, go to **Features** > **OAuth & Permissions** and scroll down to the **Bot Token Scopes** section.

4. Click **Add an OAuth Scope** and select the following scopes:
    - `app_mentions:read`
    - `chat:write`
    - `im:history`
    - `im:read`
    - `im:write`
5. Scroll up the page to the **OAuth Tokens for Your Workspace** section and click **Install App to Workspace**, and then click **Allow**.

   You should be redirected back to the OAuth & Permissions page.
6. Copy and paste your Bot user OAuth access token in the **Assistant setup** page.
7. Click **Next** to continue.

Configure your Slack bot

1. Copy the **Generated request URL**.
2. On the **Slack app settings** page, go to **Features** > **Event Subscriptions** and switch the **Enable Events** toggle `on`.
3. Paste the URL link under **Request URL**.

   Wait until the you see **Verified** with a green tick next to **Request URL**.
4. Scroll down and click **Subscribe to bot events**.
5. Select the event types you want to subscribe to. You must select at least one of the following types:
    - `message.im`: Listens for message events that are posted in a direct message channel.
    - `app_mention`: Listens for only message events that mention your app or bot.

      Note: Choose the *app_mention* entry in normal font, *not* the app_mention entry that is in bold font.
6. Click **Save Changes**.
7. On the **Assistant setup** page, click **Next**.

Connect your assistant

1. On the **Slack app settings** page, go to **Features** > **AppHome** and click **Edit** next to **App Display Name**.
2. Click **Save** once you have made the changes.
3. Switch the toggle **Always Show My Bot as Online** toggle to `on`.
4. Go to the **Show Tabs** section and switch the **Messages Tab** toggle to `on`.
5. Check the **Allow users to send Slash commands and messages from the messages tab** checkbox.
6. If you want to add support for showing buttons, menus, and disambiguation options in the Slack app, do the following steps:
    a. Go to the **Interactivity & Shortcuts** tab and enable the feature
    b. Paste your request URL in the provided text entry field.
    c. Click **Save Changes**.
7. On the **Assistant setup** page, click **Finish**.

Note: If a `token` field required for authentication is changed, then all entries in related fields must be filled and validated again.

Action considerations

The rich responses that you add to an action are displayed in a Slack channel with the following exceptions:

- **Connect to live agent**: This response type is ignored.
- **Option**: This response type shows a list of options that the user can choose from.

- After a user clicks one of the options, the existing selections disappear and replaces the selections with the user input that is generated by the user's selection. If you include multiple response types in a single response, you must position the option response type at the end to avoid confusions due to mix of responses and user inputs.
- If the options are displayed in a drop-down list, then each option value must be 75 characters or fewer in length. When a list includes 5 or more options, it is displayed in a drop-down list.

Chatting with the assistant

To start a chat with the assistant, complete the following steps:

1. Open Slack, and go to the workspace associated with your app.
2. Click the application that you created from the Apps section.
3. Chat with the assistant.

The welcome action is not processed by the Slack integration. The welcome message is not displayed in the Slack channel like it is in the assistant preview. It is not triggered from here because nodes with the `welcome` special condition are skipped in action flows that are started by users. Slack waits for the user to initiate the conversation.

The action flow for the current session is restarted after 60 minutes of inactivity (5 minutes for Lite and Standard plans). This means that if a user stops interacting with the assistant, after 60 (or 5) minutes, any context variable values that were set during the previous conversation are set to null or back to their default values.

Integrating with *SMS*

IBM Cloud

Deprecated: IntelePeer free phone number service for watsonx Assistant
Effective 31 July 2025, the IntelePeer free phone number service is discontinued for watsonx Assistant. After this date, phone numbers that use this service no longer connect, and users can't make or receive calls. To ensure continued service, migrate your phone numbers to Twilio or another telephony provider. Update your phone numbers and adjust your integration settings in watsonx Assistant. For assistance with the transition, contact support at https://www.ibm.com/mysupport/.

Add a text messaging integration so your assistant can exchange messages with your customers.

The Short Messaging Service (SMS) supports text-only messages. Typically, SMS restricts the text message length to 160 characters. The Multimedia Messaging Service (MMS) supports sending images and text messages that are over 160 characters in length. When you create a phone number with Twilio, MMS message support is included automatically. IntelePeer MMS message support is not yet available.

Customers send text messages to your hosted phone number. Twilio and IntelePeer use a messaging webhook that you set up to send a POST request with the text message body to your assistant. Each response from the assistant is sent back to Twilio or IntelePeer to be converted to an outbound SMS message that is sent to the customer. The responses are sent to the SMS provider's API for processing. You provide your SMS provider's authentication token information, which serve as your API access credentials.

Refer to the following sections to set up the integration for your SMS provider:

- Integrating SMS with Twilio
- Integrating SMS with IntelePeer

Before you begin

To integrate Twilio with your assistant, you must have at least a Developer role. For more information, see the difference in roles for Twilio.

Important: If you don't have a text messaging phone number, set up an *SMS with Twilio* account and get a phone number.

Integrating *SMS with Twilio*

1. Go to the Twilio website.
2. Create an account. Free trial accounts cannot be used for this integration.
3. From the **Develop** tab in the sidebar, click **Phone Numbers**. If **Phone Numbers** is not present, go to the Search Bar at the top and search for 'Phone Numbers', then select **Buy a number**.

4. Follow the instructions to **Buy a number**. How to search for and buy a Twilio phone number from console .

   When you get a Twilio phone number, it supports voice, SMS, and MMS automatically. Your new phone number is listed as an active number.

> **Tip:** Keep the Twilio web page open in a web browser tab so you can refer to it again later. You can also pin **Phone Numbers** to the sidebar.

Set up the integration

To set up the integration, complete the following steps:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the *SMS* tile.
3. Click **SMS with Twilio** tile.
4. Click **Confirm**.
5. From the Twilio site, click on your account name in the upper left menu to go to your account dashboard.

   Copy the following values and store them temporarily, so you can paste them into the *SMS with Twilio* integration setup page in the upcoming steps.
   - Account SID
   - Auth token
6. Return to the *SMS with Twilio* integration setup page. Click **Next** to go to Step 1 of your *SMS with Twilio* integration setup.
7. Enter your **Account SID** information. Click **Next** to go to Step 2 of your *SMS with Twilio* integration setup.
8. Enter your **Auth token** information. Click **Next** to go to Step 3 of your *SMS with Twilio* integration setup.
9. **Optional**: Enter the phone number that your Twilio account uses for SMS integration. The webhook URI is used to transfer messages, but if you add your phone number in this optional field, you can easily refer to it later. Click **Next** to go to Step 4 of your *SMS with Twilio* integration setup.
10. Copy the value from the **Webhook URI** field.

    You will add this URI to the webhook configuration in Twilio. If you want to support more than one phone number, you must add the URI to the webhook for each phone number separately.
11. Go to your Twilio account web page. From the **Develop** tab in the sidebar, click **Phone Numbers > Manage > Active numbers** .
12. From the **Active Numbers** page, click one of your phone numbers.
13. Scroll to the **Messaging** section, and then find the **Webhook** field that defines what to do when *A message comes in* .

    Paste the value that you copied from the **Webhook URI** field into it.
14. If you want to support multiple phone numbers, repeat the previous step for each phone number that you want to use.
15. Click **Save**.
16. From the **Develop** tab in the sidebar, click **Messaging > Settings > Geo permissions**. If **Messaging** is not present, go to the Search Bar at the top and search for 'Messaging', then select **SMS Geographic Permissions**.
17. From the **Messaging Geographic Permissions** page, select the country codes of the phone numbers that can text your Twilio number. By default, no country codes are allowed to text your Twilio number.
18. Return to the *SMS with Twilio* integration setup page. Click **Finish**.

Integrating with *SMS with IntelePeer*

If you don't have a text messaging phone number, set up an *SMS with IntelePeer* account and get a phone number.

1. Go to the IntelePeer website.
2. Create an account or start a free trial.

   When you get an IntelePeer phone number, it supports voice and SMS. If the number is not automatically enabled for SMS, you will need to enable it manually. Your new phone number is listed as an active number. Refer to the Atmosphere Messaging Quick Start Guide

Before you begin

To integrate IntelePeer with your assistant, you need "SMS" service with Access Type for `create`, `read`, `update`, and `delete`. For more information, see Managing Users in Customer Portal.

Set up the integration

To set up the integration, complete the following steps:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the *SMS* tile.
3. Select **SMS with IntelePeer** tile.
4. Click **Confirm**.
5. From the IntelePeer Atmosphere site, copy the **API Authentication Token** value and store it temporarily, so you can paste it into the *SMS with IntelePeer* integration setup page in the upcoming steps.
6. From the IntelePeer Customer Portal site, under the **My Applications** section, select **SMS API Management**. In the **SMS Management** title bar, click the gear icon, here you will set the SMS **Secret Input**. The Secret Input is used to prevent your webserver webhook from processing any inbound-SMS POST request that does not originate from IntelePeer. Set the **Secret Input** value here and remember it as you will need to use the value in the *SMS with IntelePeer* integration setup page in the upcoming steps.
7. Return to the *SMS with IntelePeer* integration setup page. Click **Next** to go to Step 1 of your *SMS with IntelePeer* integration setup.
8. Enter your **Account Secret** information. Click **Next** to go to Step 2 of your *SMS with IntelePeer* integration setup.
9. Enter your **Auth token** information. Click **Next** to go to Step 3 of your *SMS with IntelePeer* integration setup.
10. **Optional**: Enter the phone number that your IntelePeer account uses for SMS integration. The webhook URI is used to transfer messages, but if you add your phone number in this optional field, you can easily refer to it later. Click **Next** to go to Step 4 of your *SMS with IntelePeer* integration setup.
11. Copy the value from the **Webhook URI** field.

    You will add this URI to the webhook configuration in IntelePeer. If you want to support more than one phone number, you must add the URI to the webhook for each phone number separately.
12. Go to your IntelePeer Customer Portal site, under the **My Applications** section, select **SMS API Management**.
13. Go to the phone number(s) you want to enable for SMS. Toggle the **Enabled/Disabled** radial button beside the number and set it to *Enabled*. It may take a few minutes for the phone number to become activated for SMS.
14. Once the phone number has been enabled for SMS, you will see a webhook icon beside the number.

    Paste the value that you copied from the **Webhook URI** field into it.
15. Click **Save**.
16. If you want to support multiple phone numbers, repeat the previous step for each phone number that you want to use.

Note: For security reasons, the authentication fields are removed from view after initial setup. If a field required for authentication is changed, then all entries in related fields must be filled and validated again.

SMS Advanced configuration options

The **Advanced options** tab is available after you set up the *SMS* integration. Click the **Advanced options** tab to make any of the following customizations to the messaging behavior:

- **Initiate conversation from inbound messages**: Disable this option if you want to limit messaging support to allow messages that are sent in the context of an ongoing phone integration conversation only, and not allow customers to start a message exchange with the assistant outside of a phone call.
- **Default failure message**: Add a message to send to the customer if the SMS connection fails.
- **Base URL**: This URL is the REST API endpoint for the SMS service you are using.

Optimize your actions for messaging

For the best customer experience, design your actions with the capabilities of the *SMS* integration in mind:

- Do not include HTML elements in your text responses.

- The *SMS* integration does not support chat transfers that are initiated with the *connect_to_agent* response type.
- **Image**, **Audio**, **Video** response types allow sending a message containing media. A title and description are sent along with the attachment. Note that depending on the carrier and device of the end user these messages may not be successfully received. For a list of the supported content types for Twilio, see Twilio: Accepted Content Types for Media.

  For more information on these response types, see Response types reference.

If you want to use the same action for an assistant that you deploy to many different platforms, add custom responses per integration type. You can add a conditioned response that tells the assistant to show the response only when the *SMS* integration is being used.

For reference documentation, see *SMS* integration reference.

Troubleshooting

Find solutions to problems that you might encounter while using the integration.

- If you get a *Forbidden* message, it means the phone number that you specified when you configured the integration cannot be verified. Make sure the number fully matches the SMS phone number.

Migrating from Voice Agent with Watson

If you created an IBM® Voice Agent with Watson service instance in IBM Cloud to enable customers to exchange text messages with an assistant, use the *SMS* integration instead.

The *SMS* integration provides a more seamless integration with your assistant and supports as many phone numbers as needed. However, the integration currently does not support the following functions:

- Starting an SMS-only interaction with an outgoing text
- Configuring backup locations
- Reviewing the usage summary page. Use IBM Log Analysis instead.

To migrate from Voice Agent with Watson to the watsonx Assistant *SMS* integration, complete the following step:

1. Do one of the following things:
   - If your Voice Agent with Watson service instance uses an SMS service provider other than Twilio or IntelePeer, you cannot continue to use it. You must create an SMS account with Twilio or IntelePeer first. Complete the Before you begin - Twilio or Before you begin - IntelePeer steps to create the account. Next, set up the integration.
   - If your Voice Agent with Watson service instance uses Twilio or IntelePeer as its SMS provider, you can go directly to Set up the integration - Twilio or Set up the integration - IntelePeer.

Integrating with WhatsApp

IBM Cloud

Integrate with WhatsApp messaging so your assistant can exchange messages with your customers where they are.

Many customers use WhatsApp because it provides fast, simple, secure messaging for free, and is available on phones all over the world. WhatsApp uses the phone Internet connection to send messages so customers can avoid SMS fees.

This integration creates a connection between your assistant and WhatsApp by using Twilio as a provider.

Before you begin

To integrate Whatsapp with your assistant, you must have access to Twilio and at least a Developer role. For more information, see the difference in roles for Twilio.

1. Go to the Twilio website.
2. Create an account.
3. From the **Develop** tab, click **Phone numbers**.
4. Follow the instructions to get a phone number.

   When you get a Twilio phone number, it supports voice, SMS, and MMS automatically. Your new phone number is listed as an active number. Consider provisioning more than one phone number and going through the process of getting permission for the numbers in parallel. If your number

was used by a different business previously (because Twilio assigned you a number that was used before, for example), WhatsApp will reject it.

Tip: Keep the Twilio web page open in a web browser tab so you can refer to it again later.

Ask WhatsApp for permission to enable your Twilio number for WhatsApp

WhatsApp has a rigorous process to review all businesses that want to interact with customers over their network. WhatsApp, which is owned by Meta (formerly called Facebook), requires that you register your business with the Meta business directory.

1. To register, go to the Meta Business Tools page, and click **Create new account**. Follow the instructions to create an account.
2. Get your Meta Business Manager ID. In **Settings**, click the **Business info** tab. The Business Manager ID is at the top of the page.
3. Enable your Twilio numbers for WhatsApp using the WhatsApp Tech Provider Program web page, which is the only officially supported path by Meta to onboard your customers to WhatsApp as of January 1, 2025. For more information, see WhatsApp Tech Provider Program Overview.

   Tips for specifying the following values:

   - **Twilio Account SID**: From the Twilio site, click the home icon to go to your project dashboard to find the SID.
   - **Meta Business Manager ID**: Add the ID for the account that you created in the previous step.
   - **Do you offer self-service onboarding for your customers?** : Select **No**. By adopting the Tech Provider Program, your customers will onboard to WhatsApp using Meta's WhatsApp Embedded Signup product.
4. Click **Submit**.

Give WhatsApp time to evaluate and approve your submission. It can take up to 7 days for your request to be approved.

Set up the integration

To set up the integration, complete the following steps:

1. Go to the **Integrations** page by clicking the integrations icon ( ) in the left menu.
2. Click **Add** on the *WhatsApp with Twilio* tile.
3. Click **Confirm**.
4. From the Twilio site, click on your account name in the upper left menu to go to your account dashboard.

   Copy the following values and store them temporarily, so you can paste them into the *WhatsApp with Twilio* integration setup page in the upcoming steps.

   - Account SID
   - Auth token
5. Return to the *WhatsApp with Twilio* integration setup page. Click **Next** to go to Step 1 of your *WhatsApp with Twilio* integration setup.
6. Enter your **Account SID** information. Click **Next** to go to Step 2 of your *WhatsApp with Twilio* integration setup.
7. Enter your **Auth token** information. Click **Next** to go to Step 3 of your *WhatsApp with Twilio* integration setup.
8. Copy the value from the **Webhook URI** field.

   You can use this webhook URI to test your integration in the following section.
9. Click **Finish**.

Note: If a field required for authentication is changed, then all entries in related fields must be filled and validated again.

Testing the integration

While you wait for WhatsApp to approve your submission, you can test the integration by using the Twilio sandbox. With the sandbox, you can send and receive preapproved template messages to numbers that join your sandbox, using a shared, pre-provisioned Twilio test number.

Do not use the Twilio sandbox in production. Sandbox sessions expire after 3 days.

1. To create a sandbox, go to the Twilio Console web page and log in with your Twilio credentials. An *Activate your sandbox* prompt is displayed. Agree

to have a sandbox created, and confirm your choice.

2. Follow the instructions to create the sandbox.
3. Connect to the sandbox by sending a WhatsApp message from your device to the sandbox phone number.
4. From the **Develop** tab, click **Messaging > Settings > WhatsApp sandbox settings**.
5. In the **Sandbox Configuration** section, paste the webhook URI that you copied earlier into the *When a message comes in* field. Click **Save**.
6. You can test the integration by sending a message from WhatsApp to the shared phone number that is assigned to your Twilio sandbox.

For complete and detailed information, see Get started with the Twilio Sandbox for WhatsApp.

Finish the product integration

After WhatsApp grants permission and access to the WhatsApp network, update the integration to use your dedicated Twilio phone number instead of the sandbox number.

1. From the *WhatsApp with Twilio* integration setup page, scroll to the **Webhook** section of the **Basic setup** tab. Copy the value from the **WhatsApp webhook** field.
2. Go to your Twilio account web page and add the webhook that you copied to the Twilio configuration to complete the connection to the WhatsApp integration in Twilio.

Give customers fast access to your assistant

You can add an icon to your web page that customers can click to start a conversation over WhatsApp with your assistant.

To add an icon to your web page, complete the following steps:

1. From the *WhatsApp with Twilio* integration setup page, click the **Click to chat** tab.
2. In the **Pre-filled message** field, add text that you want WhatsApp to send to your assistant on the customer's behalf to get the conversation started.

   Specify a message that you know your assistant can answer in a useful way.
3. Copy the **Embed link** and add it to your web page. Consider adding text in front of the icon that explains what the icon does. For example, you might add a `<span>` HTML tag in front of the icon's `<span>` element that says `Have a question? Ask watsonx Assistant for help`.

   When a user clicks the icon on your web page, it opens a WhatsApp messaging session that is connected to your assistant, and adds the text you specify into the user's text field, ready to be submitted.

Action considerations

For the best customer experience, design your actions with the capabilities of the WhatsApp integration in mind:

- A text response that contains more than 1,600 characters is broken up into multiple responses.
- Do not include HTML elements in your text responses.
- The WhatsApp with Twilio integration does not support chat transfers that are initiated with the *Connect to agent* response type.
- If you use Markdown syntax, see the *Supported Markdown syntax* table.
- To include a hypertext link in a text response, specify the URL directly. Do not use markdown syntax for links. For example, specify `Contact us at https://www.ibm.com.`

Format	Syntax	Example
Italics	`We're talking about _practice_.`	We're talking about *practice*.
Bold	`There's *no* crying in baseball.`	There's **no** crying in baseball.

Supported Markdown syntax

Integrating with a custom client app

IBM Cloud

If the available integration channels do not meet your needs, you can build your own client application as the interface between the assistant and your

customers.

For more information, see Building a custom client using the API .

Integrating with Intercom

The Intercom integration is available only in the classic experience for use with a dialog skill. The Intercom integration isn't available in watsonx Assistant.

Intercom is a customer messaging platform that helps drive business growth through better relationships in the customer lifecycle.

You can integrate your assistant with an Intercom application to enable the app to seamlessly pass user conversations between your assistant and live agents.

If you integrate the assistant with Intercom, the Intercom application becomes the client-facing application for your skill. All interactions with users are initiated through and managed by Intercom.

Before you begin

To integrate your assistant with Intercom, contact your administrator for support.

One-time agent creation

You or someone in your organization must complete these one-time prerequisite steps before you add the Intercom integration to your assistant.

1. Create a functional email account for your assistant.

   Each assistant must have a valid, unique email address before it can be added to a team in Intercom.

2. From your Intercom workspace, add the assistant to your team as a new agent.

   Go to the teammate settings page in your Intercom workspace, invite the assistant as a new agent by adding the email you created in the previous step to the invite field.

   If you don't have an Intercom workspace set up yet, create one at www.intercom.com. At a minimum, you need a subscription to the *Inbox* product from Intercom to be able to create a workspace.

3. From the assistant email account you created earlier, find the invitation from Intercom. Click the link in the email to join the team. Sign up using the assistant's functional email address, and then join the team.

4. **Optional**: Update the profile for your assistant.

   You can edit the name and profile picture for your assistant. This profile represents the assistant in private agent communications within your workspace, and in public interactions with customers through your Intercom apps. Create a profile that reflects your brand.

   Click the Intercom profile icon in the navigation bar to access profile and workspace settings.

Preparing the dialog

Complete these steps in your dialog skill so the assistant can handle user requests, and can pass the conversation to a live agent when a customer asks for one.

1. Add an intent to your skill that can recognize a user's request to speak to a live agent.

   You can create your own intent or add the prebuilt intent that is named `#General_Connect_to_Agent` that is provided with the **General** content catalog that is developed by IBM.

2. Add a root node to your dialog that conditions on the intent that you created in the previous step. Choose **Connect to human agent** as the response type.

3. Prepare each dialog branch to be triggered by the assistant from the Intercom app.

   Every root dialog node in the dialog can be processed by the assistant while it is functioning as an Intercom teammate, including root nodes in folders. You specify the action that you want the assistant to take for each dialog branch later when you configure the Intercom integration. Therefore, although you cannot hide a node from Intercom, you can configure the assistant to do nothing when the node is triggered.

   Complete the following fields of the root node of each dialog branch:

   - **Node name**: Give the node a name. This name is how you identify the node when you configure interactions for it later. If you don't add a name, you must choose the node based on its node ID instead.

- **External node name**: Add a summary of the purpose of the dialog branch. For example, *Find a store*.

  This information is shown to other agents on the assistant's team when the assistant offers to answer a user query. If there is more than one node that can address the query, the assistant shares a list of responses with live agents to get their advice about which response to use.

  > **Tip:** Do **not** add an external node name to the root node that you created in Step 2. When an escalation occurs, your assistant looks at the external node name of the last processed node to learn which user goal was not met successfully. If you include an external node name in the node with the connect to live agent intent, then you prevent your assistant from learning the last goal-oriented node that the user interacted with before they escalated the issue.

4. If a child node in the branch conditions on a follow-up request or question that you do not want the assistant to handle, add a **Connect to human agent** response type to the node.

   For example, you might want to add this response type to nodes that cover sensitive issues only a live agent can handle or that track when an assistant repeatedly fails to understand a user.

   At run time, if the conversation reaches this child node, the dialog is passed to a live agent at that point. Later, when you set up the Intercom integration, you can choose a live agent as a backup for each branch.

Your dialog is now ready to support your assistant in Intercom.

Dialog considerations

Some rich responses that you add to a dialog are displayed differently within the "Try it out" pane from how they are displayed to Intercom users. The following table describes how the response types are treated by Intercom.

Response type	How displayed to Intercom users
Option	The options are displayed as a numbered list. In the **title** or **description** field, provide instructions that explain to the user how to choose an option from the list.
Image	The image **title**, **description**, and the image itself are rendered.
Pause	Whether or not you enable it, a typing indicator is not displayed during the pause.

Response types

For more information about response types, see Rich responses.

Adding an Intercom integration

1. From the Assistants page, click to open the assistant tile that you want to deploy.
2. From the Integrations section, click **Add integration**.
3. Click **Intercom**.

   Follow the instructions that are provided on the screen. The following sections help you with the steps.

Connecting the assistant to Intercom

As soon as you give Intercom permission to use the assistant, the assistant becomes a viable member of the Intercom team.

Live agents can assign messages to the assistant by using Intercom's assignment rules. Messages can be assigned to the assistant in the following ways:

- Automatic assignment of inbound conversations to a teammate or team inbox based on some criteria
- Manual reassignment is made by a live agent at run time.

1. When your dialog is ready, click **Connect now**.
2. Click **Access Intercom** to be redirected to the Intercom site.

   > **Important:** Log in to Intercom by using the assistant's functional email address and password, not your own. You want to establish the connection to a functional ID that is shared by more than one person in your organization.

3. Click **Authorize access**.
4. Click **Back to overview**.

Configuring message routing

Assign teammates as backups for the assistant in case the assistant needs to transfer an in-progress conversation to a live agent. You can choose a different team or teammate to be the backup contact for each dialog branch.

To set up routing assignments for escalations from the assistant to a live agent, complete the following steps:

1. From the Intercom integration page, click **My Dialog Skill is ready** to confirm that you prepared your dialog, if you completed the Preparing the dialog procedure.
2. In the *Settings* section, click **Manage rules**.

   If you make no changes, the backup contact for all of the nodes remains unassigned.
3. Click **New rule**.
4. From the *Choose node* list, choose the node for the dialog branch you want to configure.

   Remember, branches are identified by their node name. If you did not specify a node name, then the node's ID is displayed instead.
5. Choose the team or live agent to be the backup contact for this dialog branch. The user query is escalated to this person if the assistant cannot answer the query or uses a child node with a *Connect to human agent* response type.
6. To define routing rules for other dialog branches, click **New rule** again, and repeat the previous steps.

   Don't forget to set up an assignment for any root nodes that have a *Connect to human agent* response type in a child node in their branch. If you do not transfer the associated root node to a specific person or team, a sensitive matter can be transferred to the *Unassigned* inbox.
7. After you add rules, click **Return to overview** to exit the page.

Give the assistant permission to monitor and answer user queries

When you want the assistant to start monitoring an Intercom inbox, and answering messages on its own, turn on monitoring.

Your assistant watches user inquiries as they are logged in Intercom. When the assistant is confident that it knows how to answer a user query, the assistant responds to the user directly. (The assistant is confident when the top intent that is identified by your assistant has a confidence score of 0.75 or higher.)

If you do not want the assistant to answer certain types of user queries, then you can add rules to specify other actions for the assistant to take per dialog branch. For example, you might want to start incorporating the assistant into the team conservatively, allowing the assistant to suggest responses as it transfers user messages to other teammates. Over time, after the assistant proves itself, you can give it more responsibility.

To configure how you want the assistant to handle specific dialog branches, define rules.

1. From the Intercom integration page, in the *Enable your assistant to monitor an inbox* section, switch monitoring **On**.
2. In *Settings*, click **Manage rules**.

   If you do not define rules, the assistant is configured to monitor the *Unassigned* inbox and automatically answer user inquiries that it is confident it can address.
3. From the *Monitor Intercom inbox* field, choose the Intercom inbox that you want the assistant to monitor.
4. Click **New rule** to define a unique interaction pattern for a specific dialog branch.
5. From the *Choose node* list, choose the node for the branch you want to configure.

   Remember, branches are identified by their node name. If you did not specify a node name, then the node's ID is displayed instead.
6. Pick the type of action that you want the assistant to do when this dialog node is triggered. The action type options are:
   - **Do nothing**: The assistant is not involved in the response. The user's message remains in the inbox for someone else to address.
   - **Send to team or teammate**: The assistant evaluates the user input to determine its goal, and then transfers it to the appropriate teammate.
   - **Suggest to team or teammate**: The assistant provides the teammate with suggestions for how to respond by sharing notes with the live agent through the internal Intercom app.

- If the user input triggers a dialog branch, meaning a root dialog node with child nodes that represents a comprehensive interaction, then the assistant indicates that it can address the request, and offers to do so.
- If the user input triggers a root node with no children, then the assistant shares the programmed response with the live agent, but doesn't respond directly to the user.
- If the input triggers more than one dialog node with high confidence, the assistant provides the live agents a list of possible responses and asks the teammate to choose the best response.

In every case, the live agent decides whether to let the assistant take over.

- **Answer**: The assistant responds to the user directly, without conferring with any other teammates.

Teammate involvement is required for any nodes to which you assign the *Send to team or teammate* or *Suggest to team or teammate* action types. Be sure to go back and add a rule that assigns the right person or team as the backup for this dialog node in particular.

7. To define unique interaction settings for other dialog branches, click **New rule** again, and repeat the previous steps.
8. After you add rules, click **Return to overview** to exit the page.

Testing the integration

To effectively test your Intercom integration from end-to-end, you must have access to an Intercom application. The workspace must have an associated user interface client. If it does not, see Build your app for help.

Submit test user queries through a client application that is associated with your Intercom workspace to see how the messages are handled by Intercom. Verify that messages that are meant to be answered by the assistant are generating the appropriate responses, and that the assistant is not responding to messages that it is not configured to answer.

Improving your assistant

Use analytics to review your entire assistant at a glance

The **Analyze** page provides a summary of the interactions between users and your assistant.

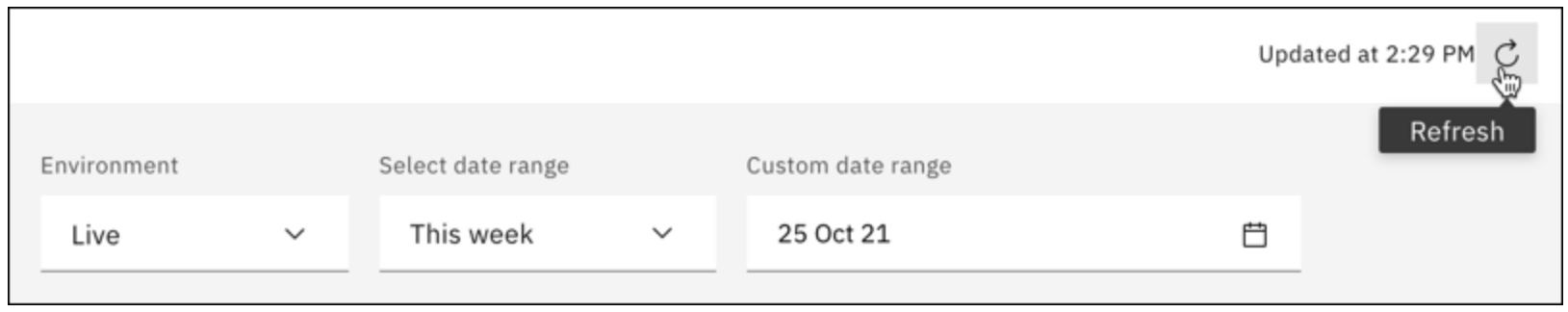
Analytics help you to understand the following things:

- *What do customers want help with today?*
- *Is your assistant understanding and addressing customer needs?*
- *How can you make your assistant better?*

To see analytics information, select **Analyze** in the navigation bar.

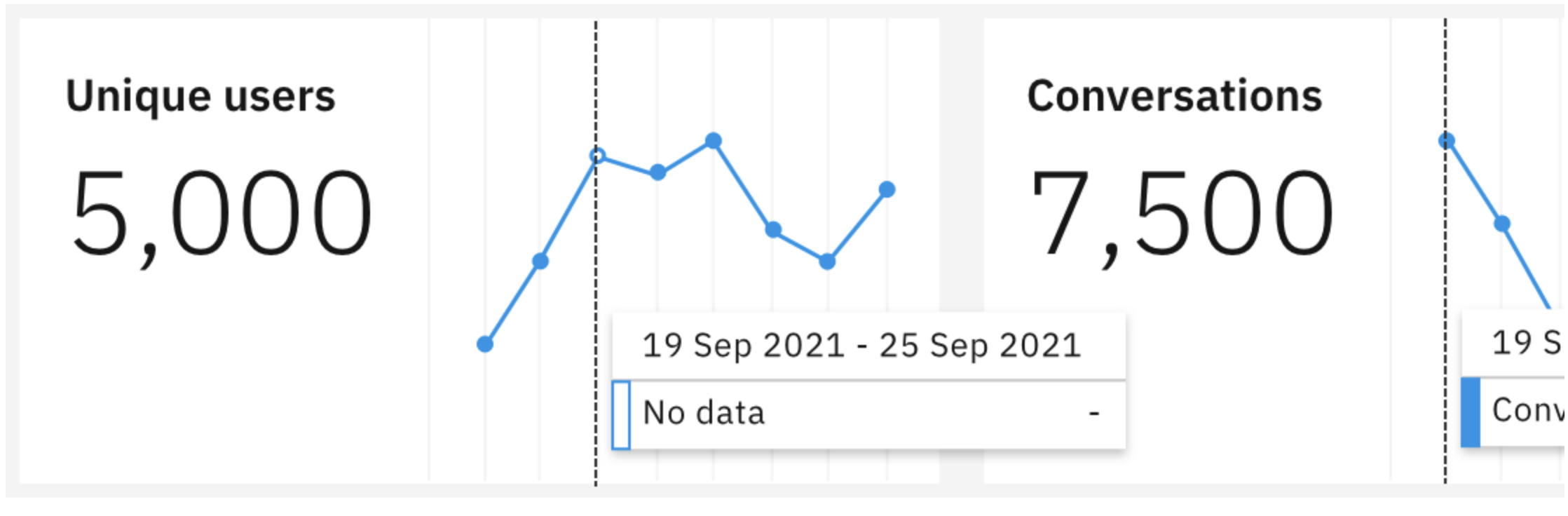
> **Note:** You can search and analyze the conversational search requests from the customer in the **Analyze** page.

Choosing the environment and date range



Unique users, conversations, and user requests

These three traffic metrics -- *unique users*, *conversations*, and *user requests* -- provide you with data about the volume of user engagements with your assistant.



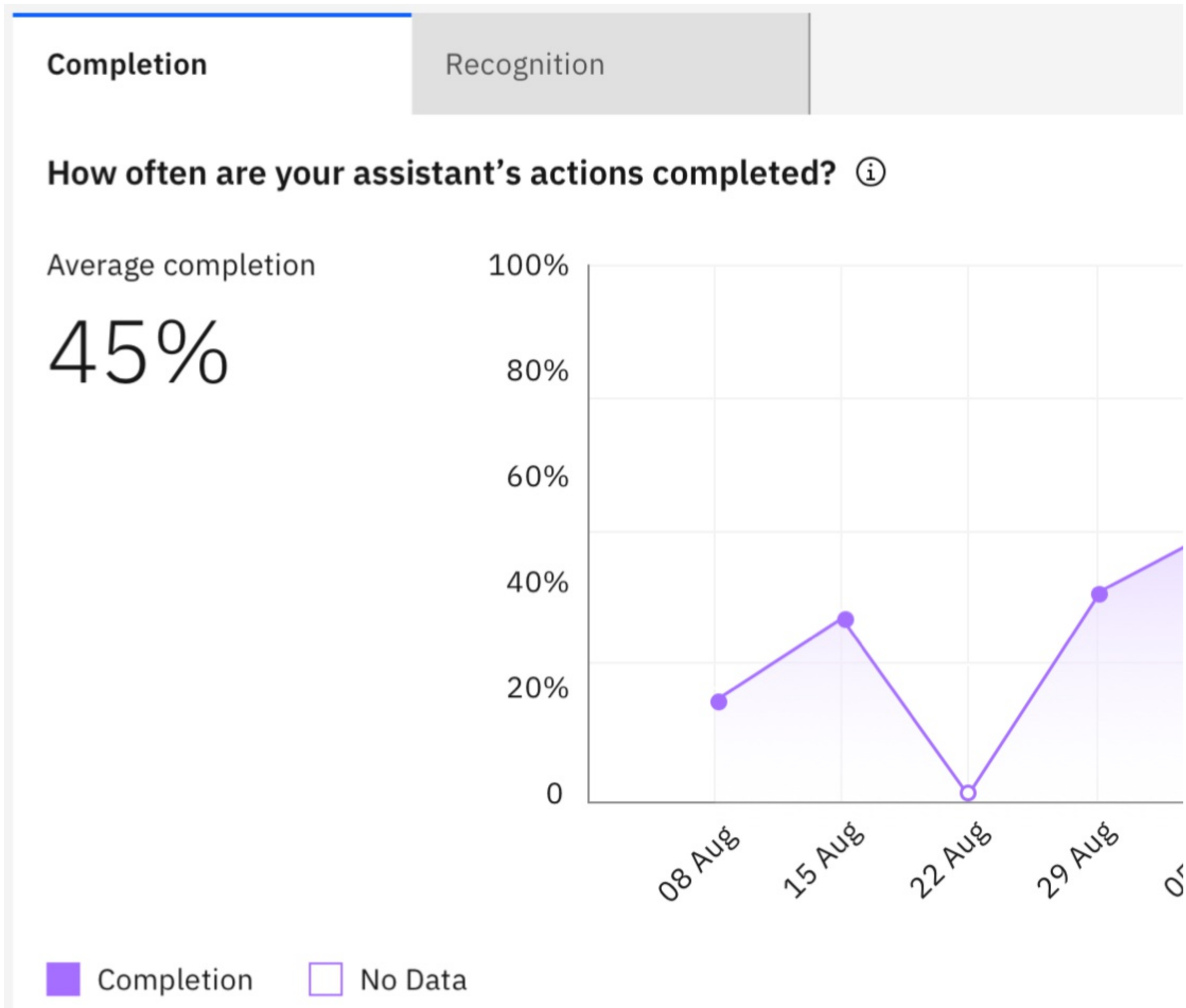
Unique user is anyone who interacts with your assistant. *User ID* identifies each user, using a unique label to track the level of service usage. A unique user can have multiple conversations, but a conversation never has more than one unique user.

Conversation is a set of messages consisting of the messages that an individual user sends to your assistant, and the messages your assistant sends back. Conversations can have multiple requests within a single conversation, but a single request doesn't span more than 1 conversation.

Request is a root-level utterance, such as an initial question or request, that signals the start of a specific flow. A user can initiate multiple requests. Requests are meant to represent the core concepts or topics your users are asking about. A request can have multiple steps within it, for example `I want to order a pizza` is a request. `Delivery/takeout`, `Small/Medium/Large`, `Cheese/Pepperoni/Mushrooms/Peppers` are all steps within the request of ordering a pizza.

Completion and recognition

The *completion* and *recognition* charts provide information about the actions in your assistant.



Completion

Completion is the measurement of how often users are able to successfully get through all steps of an action.

Your assistant measures when someone reaches the final step of an action. The completion chart provides an overview of all the actions you have built and how many of these are being completed or not.

Completion is only applicable when a user question or request matched to an action, and the action starts.

One action can be triggered multiple times, so to better understand individual action performance, click *Action completion* to understand each action in more detail.

Recognition

Recognition is the measurement of how many requests are being recognized by the assistant and routed into starting an action.

The recognition chart provides you with a view into how many requests are matched to actions. This helps you to understand where you may have content gaps, where you might want to build new actions, or how existing actions aren't matching properly to user requests.

Customer requests are considered unrecognized if:

- The request triggers the **No matches** action
- The assistant asks a clarifying question and the customer chooses `None of the above`

To get more detail, click on *Unrecognized requests* to review the requests that are not being recognized by the assistant, so you can create new actions that address questions and issues that aren't being answered.

Most frequent, least frequent, and least completed

These most/least cards help you to quickly identify specific actions that might need your attention. From these lists, you can click a specific action to do more analysis.

Most frequent actions shows the actions that have the most recognized requests matched with that action.

Least frequent actions is the exact opposite of *most frequent*. It can help you identify actions that possibly have poor training and are not matching well to actual requests from your users.

Least completed actions shows a list of actions with the lowest completion rate, by percentage. This measurement doesn't take traffic into account. This can be another good indication of actions where maybe the flow or content can be improved, and you can spend time investigating.

Conversational search analytics

Plus Enterprise IBM Cloud Pak for Data IBM Software Hub

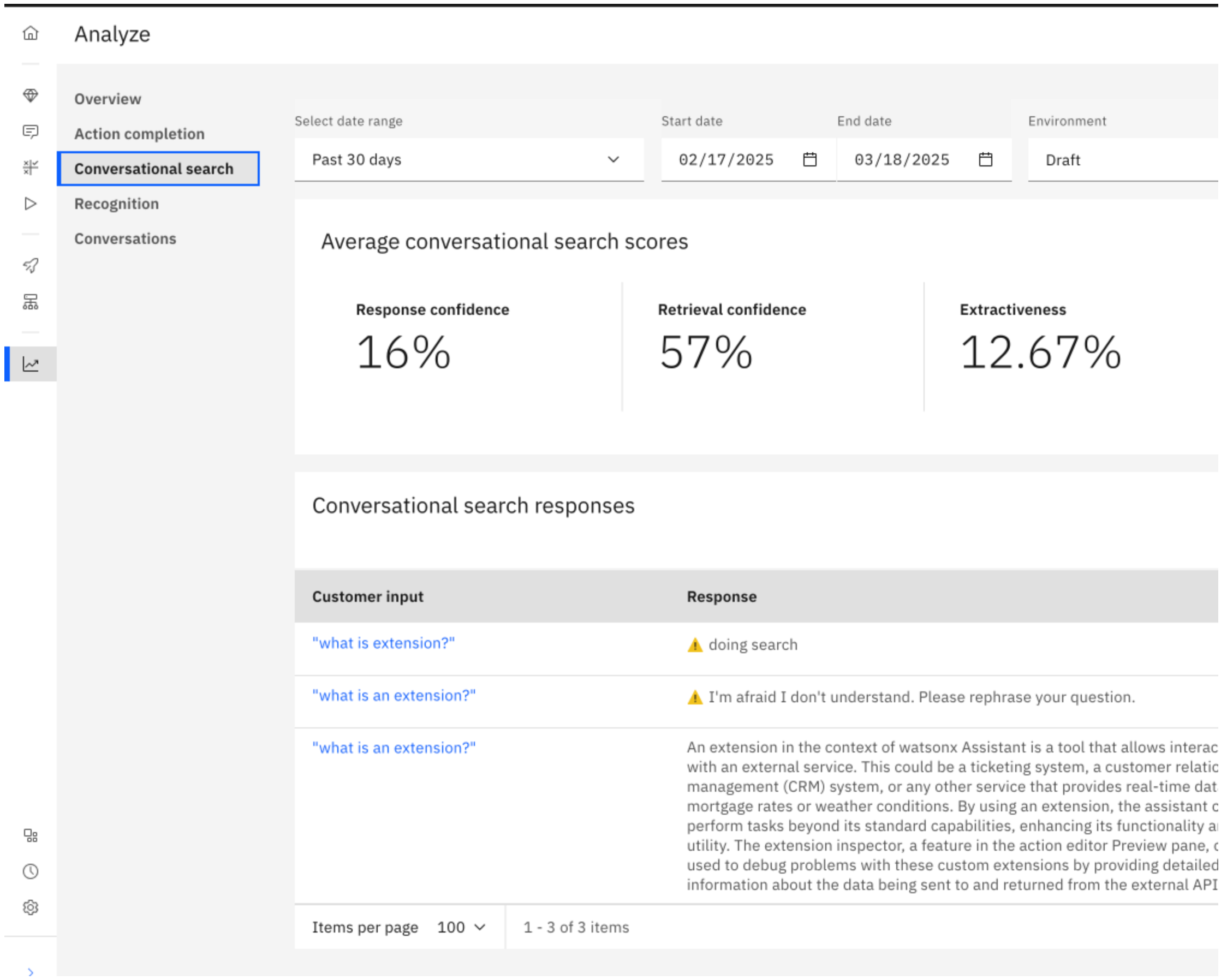
Overview

You can analyze the performance of your conversational search by using a holistic routing graph of your assistant. In the watsonx Assistant home page, go to **Analyze** > **Conversational search** to open the conversational search statistics as a preview.

Analyzing data and Conversational search scores

You can see the average scores for citations per response, answer length, answer confidence, and extractiveness in the draft configuration. You can filter conversational search responses that use successful conversational search responses or “I don’t know”. Click any **Customer input** to view the inline citations for that conversational search.

Hover on the information icon (ⓘ) next to the customer input to see the query text inferred from the context.



For a single utterance-response pair, you can view the following metrics:

Response confidence score

The response confidence score is the estimated probability that the assistant’s response is correct, relevant, and useful in addressing the user's query or request for the available content.

Retrieval confidence score

The retrieval confidence score measures how certain the system is that it retrieved the most relevant information from its database to answer a user's query. It is the estimated probability that the retrieved data contains the necessary details to respond accurately to the user's request.

Extractiveness

Extractiveness is the extent to which the response is directly derived from the input. It is the fraction of the response that consists of sequences of words that are in the search results. A high score indicates that much of the response is directly quoted from the sources. A low score indicates that the response is abstracted or paraphrased from the sources. However, it can also mean that the response is not supported by the sources.

Citations

Citations refer to the acknowledgment of the sources of data, models, or algorithms that the system uses to generate its outputs or make its predictions. On the analytics page, you can see the number of citations that are associated with the response.

Response length

The number of characters in the response.

Average citations per response

The average number of citations that are received by each response that is provided by the assistant.

Average response length

The average length of characters required to provide a helpful response.

> **Note:** For all the metrics, the average is the average among all questions for which we generated a response, regardless of whether we provided it or not.

Understand your most and least successful actions

The **Action completion** page of watsonx Assistant provides an overview of how all your assistant's actions are doing. You can:

- Understand how well users are progressing through the action steps
- Identify problem areas within actions
- Investigate why users are having issues where they might escalate to a live agent, start a new action, get stuck on a step, or stop the conversation

Definition of completion

Completion measures how often within a time period that a user reaches the end step of an action.

Reasons for completion

An action is considered complete when:

- A final (end) step is reached
- Search reached a final step
- The action called another action with the `End this action after the other action is completed` option, and the other action is finished.
- Connect to agent transfer occurs according to the step response and without involving the Fallback action
- The last step of the action is finished and there are no more steps

Reasons for incompletion

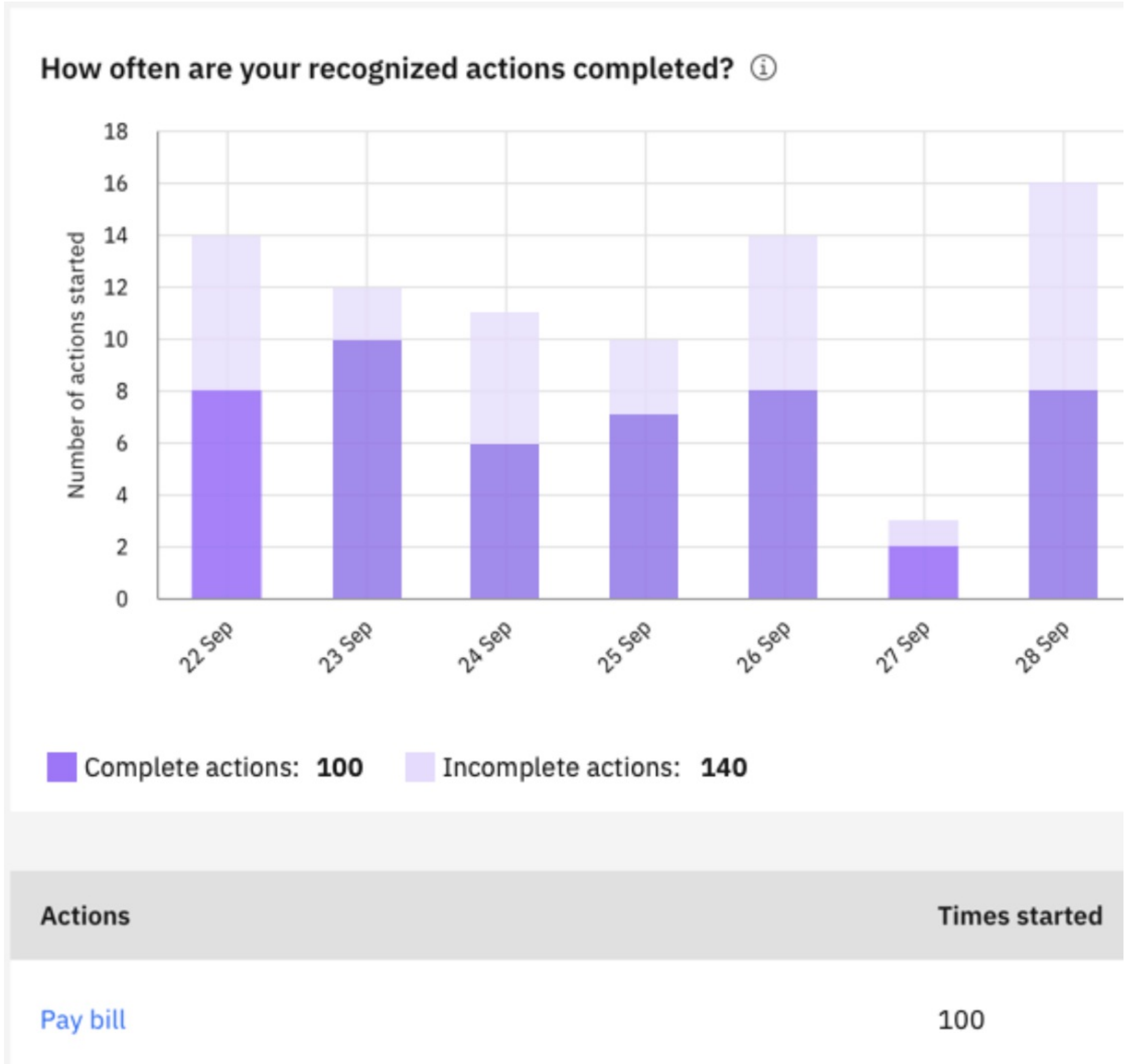
An action is considered incomplete for these reasons:

Reason	Description
Escalated to agent	The user explicitly asks to speak to someone, triggering the Fallback action. For more information, see When your customer asks to speak to a live agent. Or, the user chooses live agent escalation from the list of suggestions in web chat.
Started a new action	The user changes the topic of the conversation, triggering another action, and either doesn't return to the original action or the other action is also incomplete.
Stuck on a step	Triggered during step validation where a user exceeds the maximum retries for the particular step. Default tries is 3, but you can change this setting. For more information, see Customizing validation for a response.
Abandoned	An action is considered abandoned if it was not completed after 1 hour of inactivity and doesn't meet the criteria for any other incompletion reason.
Trigger phrase	During an action, the customer used a keyword or phrase that activated the *Trigger word detected* action, leaving the original action unfinished. For more information, see Detecting trigger words.

Incompletion reasons

Improving completion

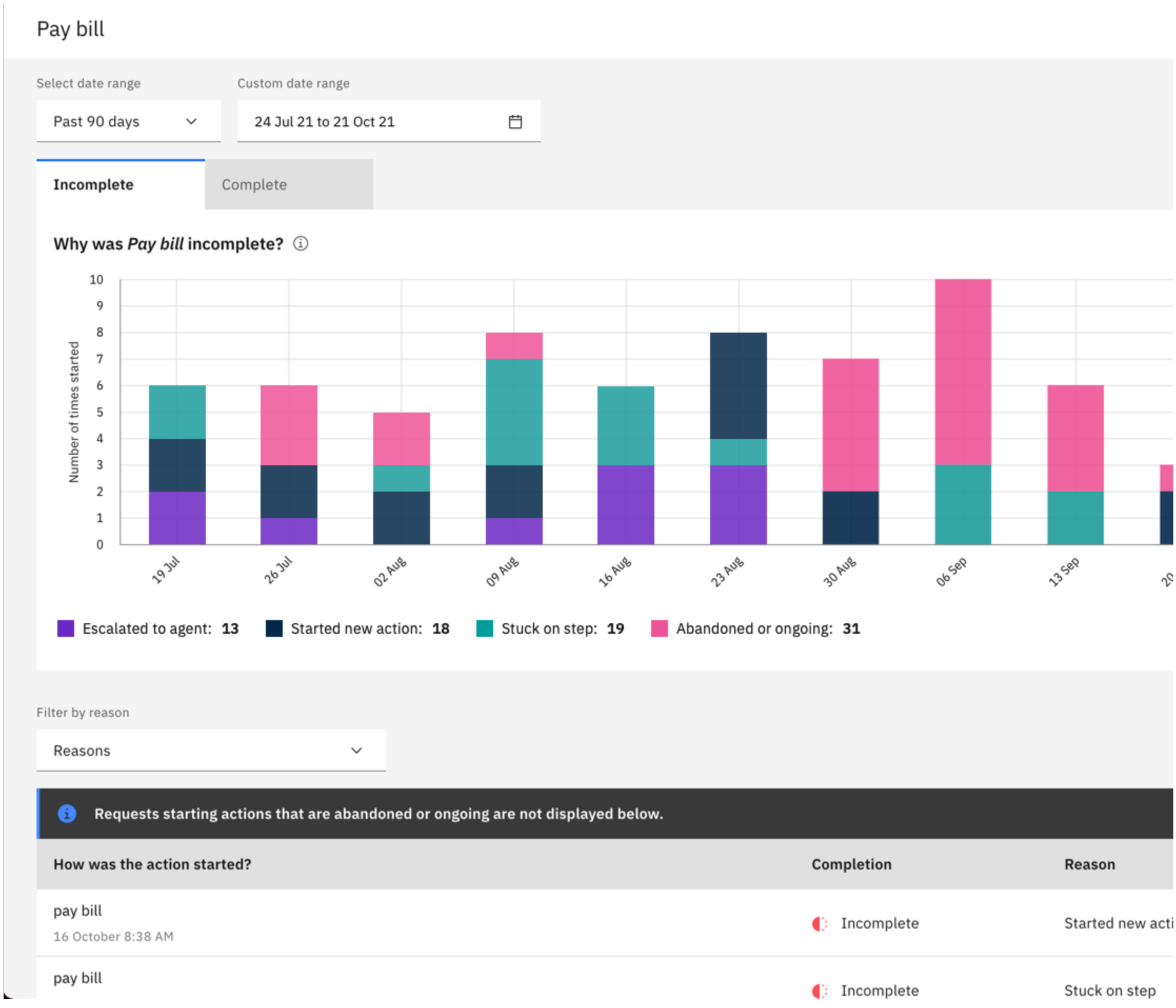
To help you identify actions that need improvement, the **Action completion** page is organized into a **How often** chart and an **Actions** table. The **How often** chart shows either the percentage of complete actions or the number of complete and incomplete actions during a time frame. You can toggle between a line chart that shows the percentage of complete actions or a bar chart that shows the number of complete and incomplete actions. The **Actions** table provides number of requests, total incomplete, and completion percentage per action.



Actions	Times started
Pay bill	100
Deposit money	40
Withdraw Action	100

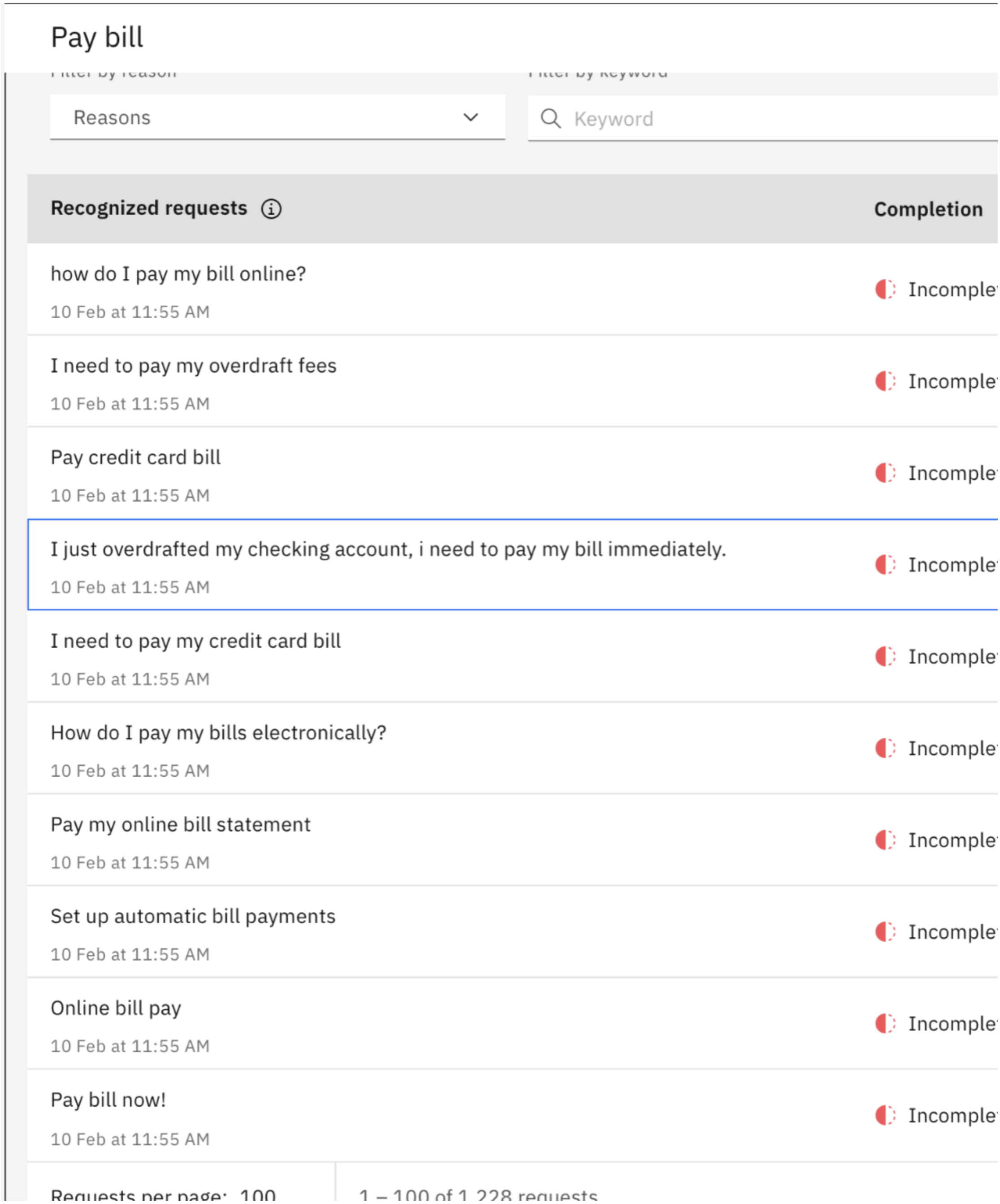
You can use the **Actions** table to focus on improving the completion of your actions. **Incomplete** results provide a focus area where you can make the biggest impact. The actions with the most incompletions are where most users can't get their questions answered or requests resolved.

To work on a specific action, click the action name in the table. A page opens with completion details for that single action.



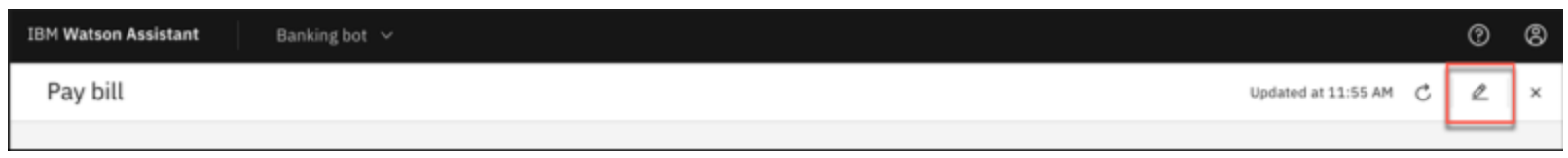
The *Incomplete* and *Complete* tabs provide details by time frame. The *Incomplete* tab bar chart shows data for the four incompletion reasons. The *Complete* tab bar chart is organized by conversations that are either completed by the assistant or completed by a planned `connect to agent` response.

As you click each tab, a table shows the list of either incomplete or complete requests. To explore individual requests in detail, you can click each one. A window opens showing the full back and forth between your customer and the assistant, including step interactions. The window also provides a summary of how many requests there were, how many were recognized, whether search was initiated, and the duration of the conversation.



Editing an action

When you decide on changes to improve the action, click the edit icon in the header for the single action, which opens your action for further work on its steps.



Use unrecognized requests to get action recommendations

IBM Cloud | Plus | Enterprise

Use the **Recognition** page to analyze unrecognized requests. You can use this information to create new actions that address questions and issues that aren't being answered by your assistant.

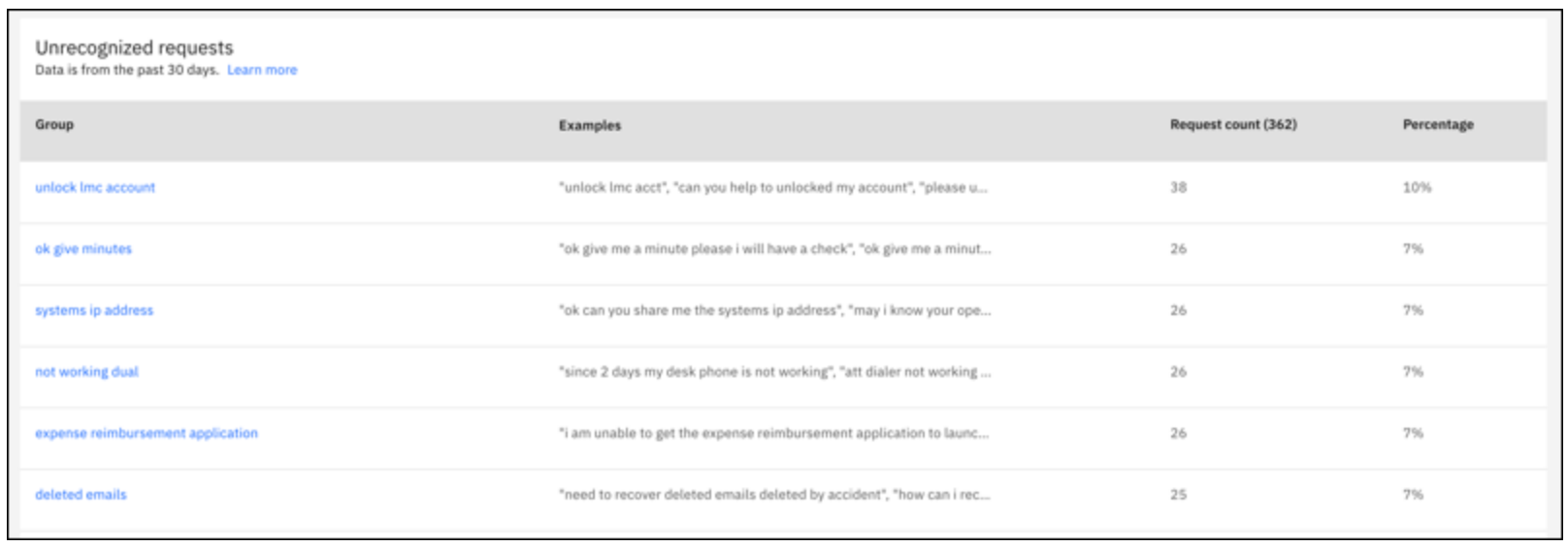
> **Note:** You can analyze the unrecognized requests in assistants that you created for the English, Spanish, and Brazilian Portuguese languages if you are in the Plus or higher plans of watsonx Assistant.

Recognition measures the requests within a time period that are recognized and successfully routed to an action. Customer requests are considered unrecognized if:

- The request triggers the **No matches** action
- The assistant asks a clarifying question and the customer chooses `None of the above`

Viewing groups of unrecognized requests

You can view groups of unrecognized requests for the draft or live environment. IBM® watsonx™ Assistant generates groups of similar unrecognized requests from the last 30 days so that you can decide whether to add the requests as example phrases to a new action.



Unrecognized requests

The algorithm that generates groups considers several factors in creating the groups:

- Unrecognized requests that have fewer than 2 or more than 35 significant words are removed from consideration. Common words such as `my` or `is`, or punctuation such as `?`, are not considered significant. Phrases that are too short or too long are usually not effective as example training phrases for your assistant.
- The unrecognized requests are compared to the latest version of your actions so that requests that would no longer be unrecognized with your latest actions version are filtered out.
- Groups for which the request count is less than 10 are excluded. But if this might result in less than 5 groups, the algorithm tries to produce groups that include more than 5 request counts until a total of 5 groups are produced.
- If you have a small volume of data, the algorithm allows examples closer to the existing training data for grouping.

As a result of the algorithm:

- A list of groups might not always appear
- The groups might not include all unrecognized requests that you see on the **Conversations** page

Several events cause your groups to be refreshed by using the latest data:

- The first time that you visit the **Recognition** page for a specific environment, watsonx Assistant generates groups
- If it has been at least 1 day since the groups were last generated, or if you edit your actions in your draft environment, the groups are refreshed when you return to the **Recognition** page
- You can refresh the groups by using the **Refresh** icon

This table explains the details of the list of groups:

Column	Description
Group	A name is generated based on the example phrases in the requests.
Examples	Some of the unrecognized requests are shown as a preview.
Request count	The number of unrecognized requests in the group. Focus on groups with a higher count of unrecognized requests.
Percentage	The percentage of unrecognized requests relative to the other groups. Focus on groups with a higher percentage of unrecognized requests.

Unrecognized request groups list

Creating actions from unrecognized requests

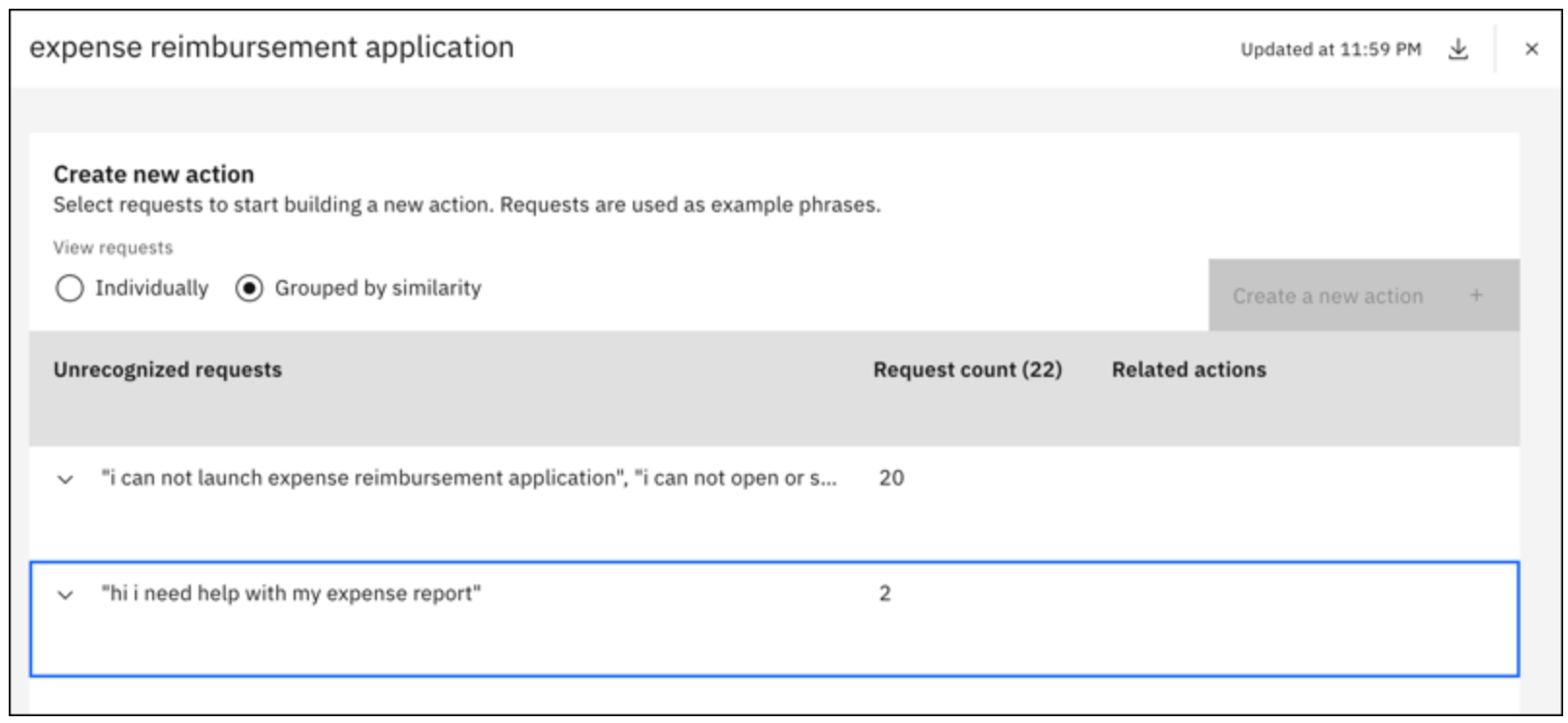
To create actions based on unrecognized requests:

1. Click a group to open its page. Each unrecognized request is listed individually.

Column	Description
Unrecognized requests	Verbatim requests from customer conversations
Request count	The number of times the unrecognized request is included in this group
Related actions	Existing actions that you might modify to address the unrecognized request

Unrecognized request group page

2. If watsonx Assistant identifies more similarities among the examples, you can click **Grouped by similarity** to further categorize the list.



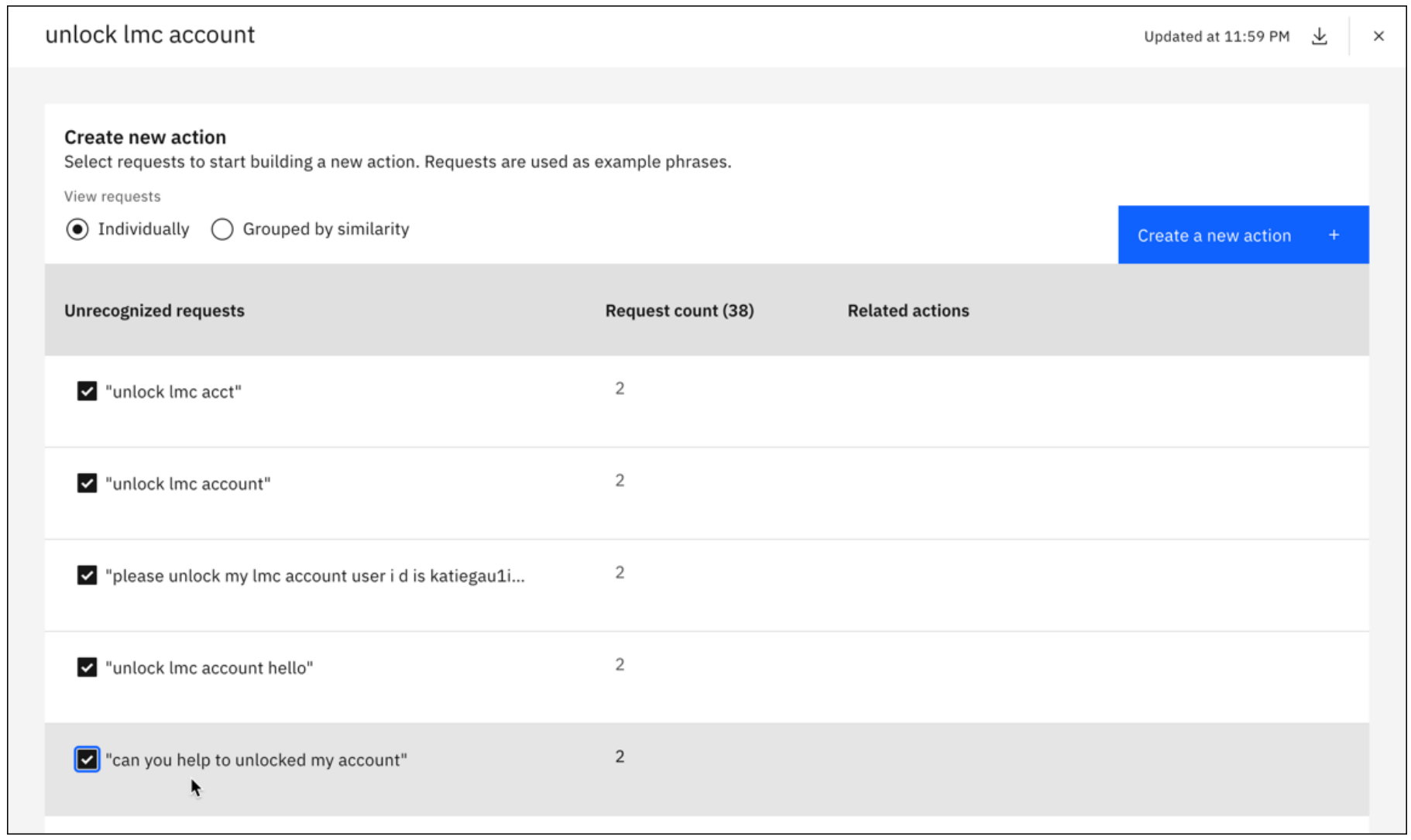
Grouped by similarity

In a group named `expense reimbursement application`, requests might be further grouped by similarity. For example:

- Problems logging in to or starting the expense reimbursement application
- Issues with crashes of the expense reimbursement application
- Requests for help or assistance with the expense reimbursement application

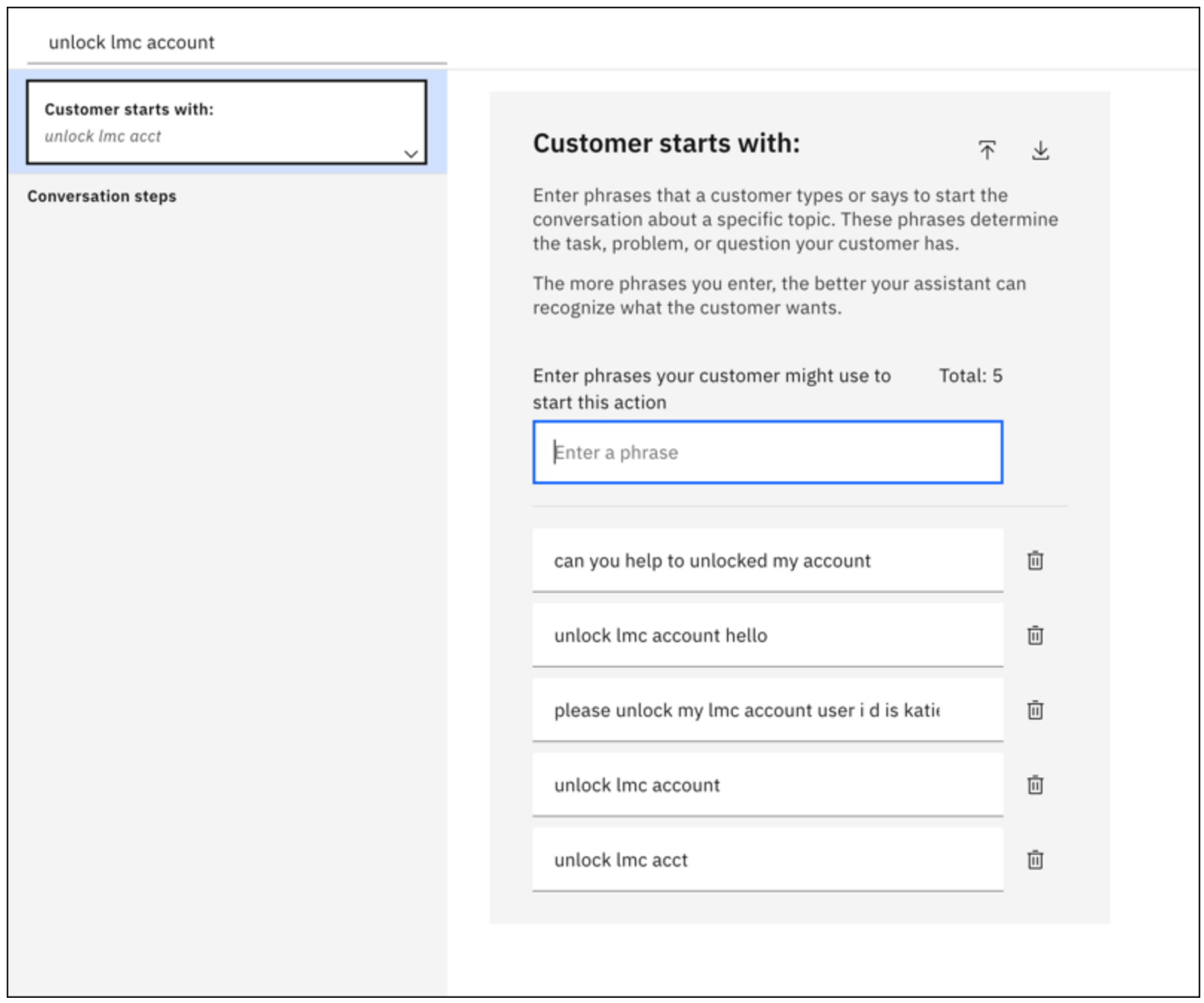
These similarities can help you decide to add one or more actions that correspond to what you want to cover with your assistant, rather than adding all requests to a single action.

3. You can click to select unrecognized requests that you want to use as example phrases in a new action.



Select requests

4. After you make your selections, click **Create new action**.
5. Enter a name for the action, or use the default, and then click **Apply**.
6. The action editor opens with each of the unrecognized requests that are included as an example phrase. You can now build an action to address these questions or issues.



Customer starts with

Modifying existing actions

You can also use the unrecognized request groups to identify existing actions that you might want to modify. You can:

- Add unrecognized requests as example phrases to existing actions
- Move example phrases from existing actions and add them to any new actions you create based on unrecognized requests

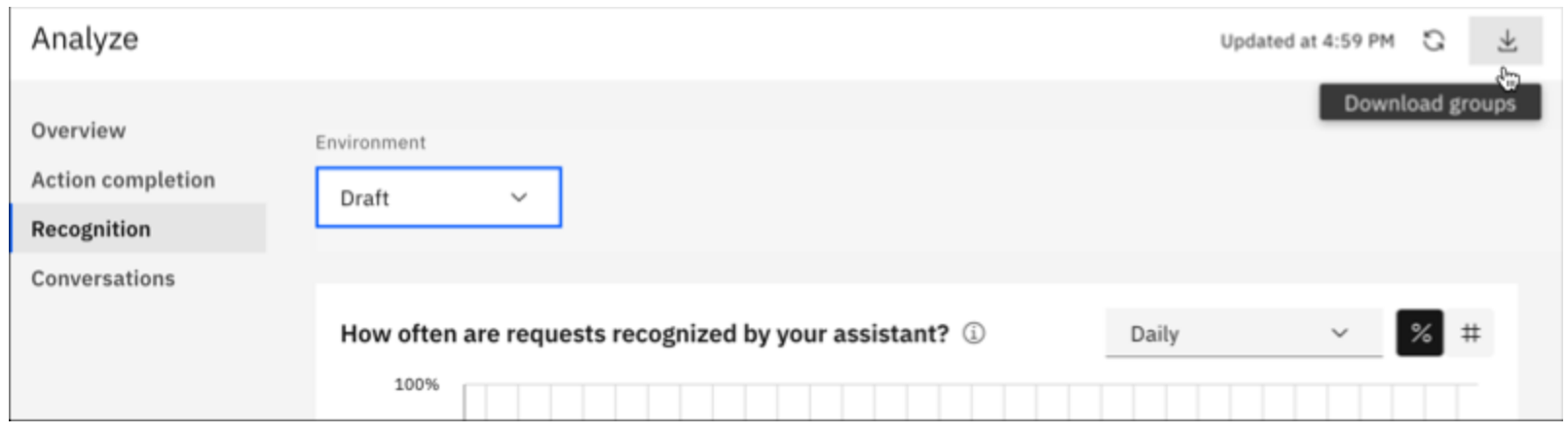
If a group lists related actions, you might focus on modifying them to address the unrecognized request.

Related actions starts

Downloading groups

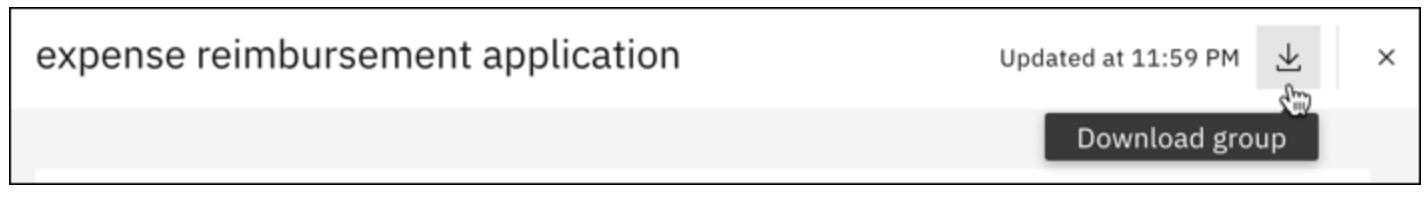
You can download all group data or individual group data in a CSV file.

To download all group data, click the **Download groups** icon on the **Recognition** page.



Download groups

To download data for an individual group, open the group, then click the **Download group** icon.



Download group

The CSV file includes this information:

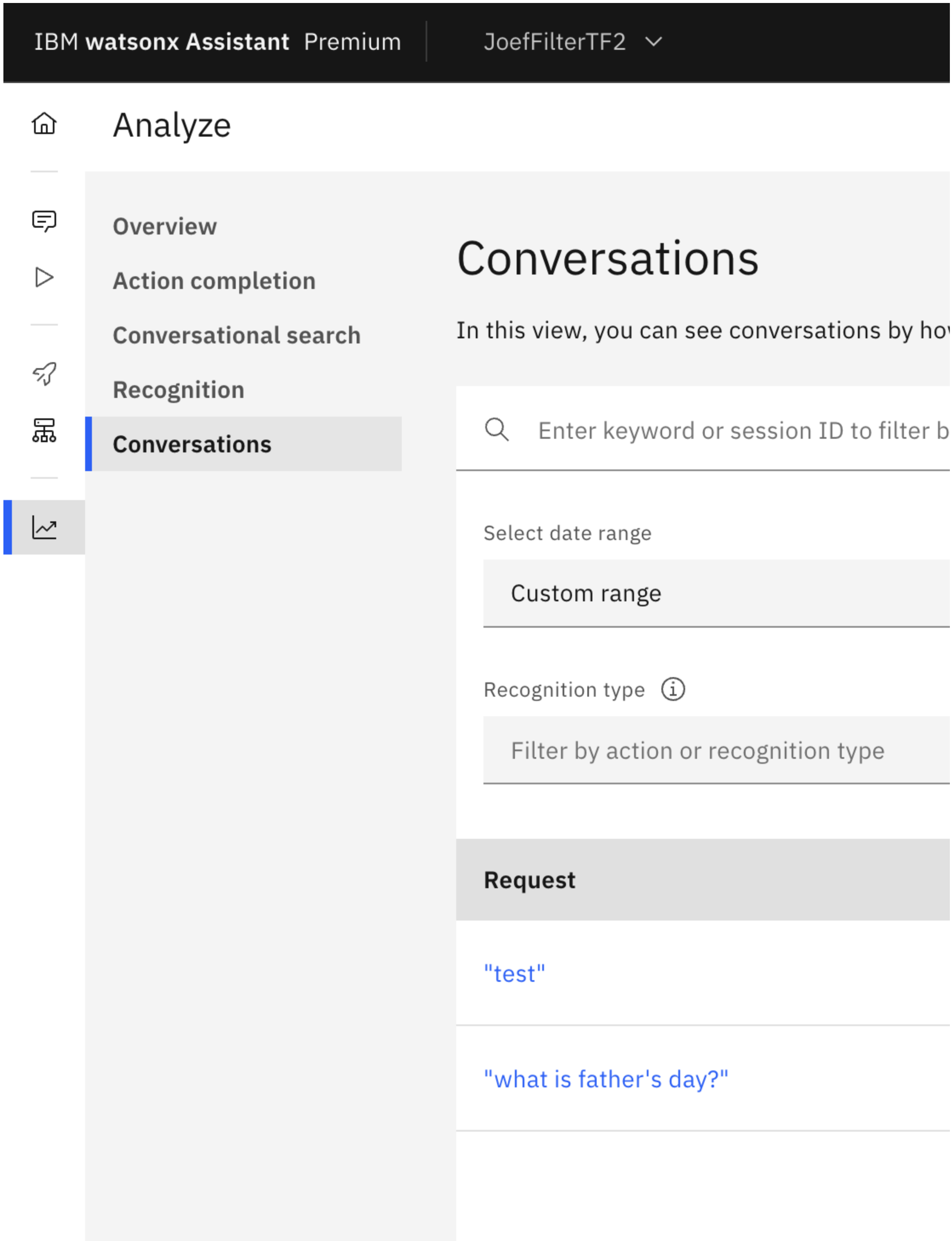
Column	Description
Group	The name of the unrecognized requests group
Example	Each verbatim request in the group
Count	The number of times the unrecognized request is included in this group
Group Id	ID number for the group
Similarity Group Id	ID number for any further grouping by similarity, for example, 1, 2, and 3 if there are three similarity groups
Example Id	ID number for the example within the group or similarity group

Download group CSV file

Review customer conversations

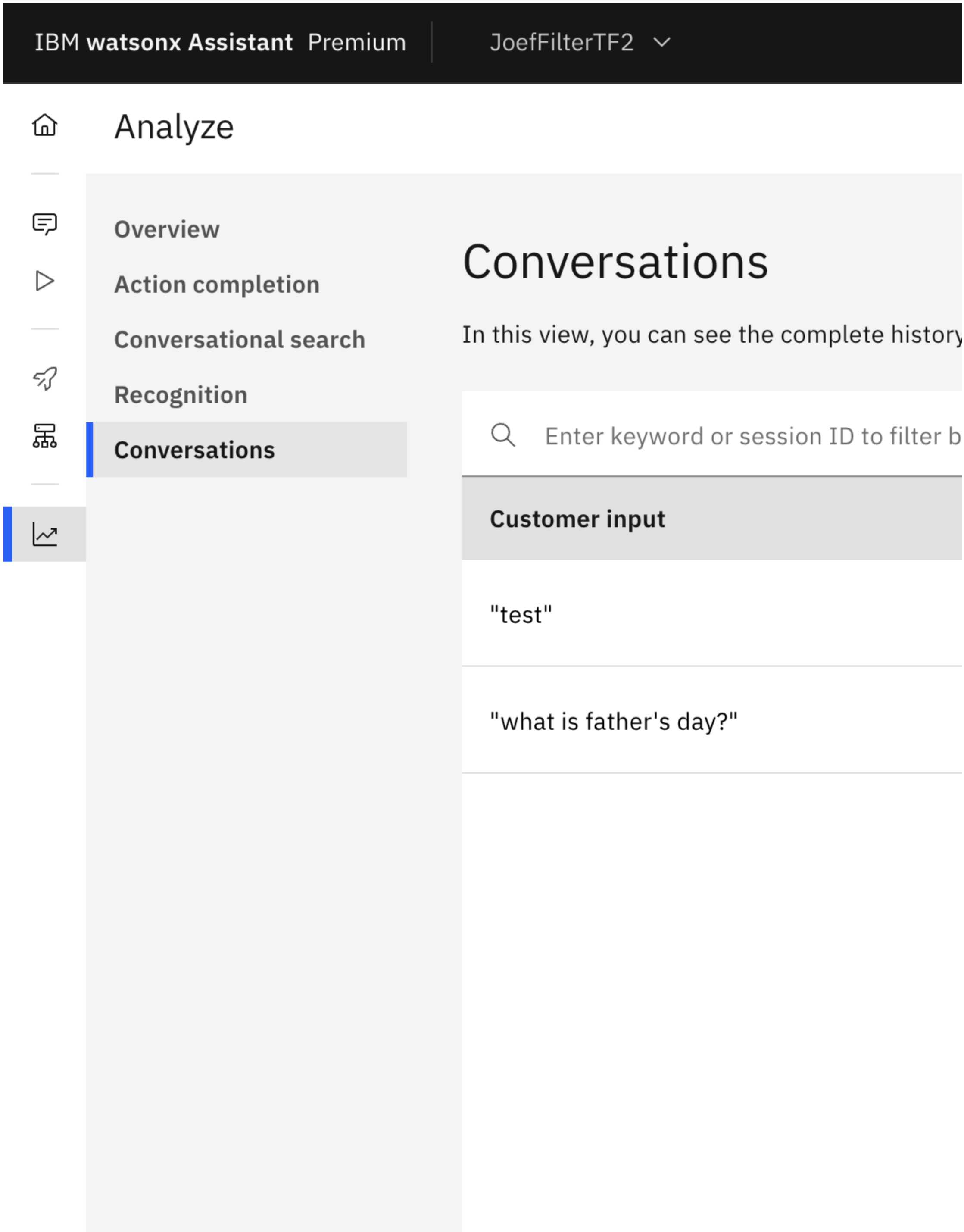
The **Analyze** page of watsonx Assistant provides a history of conversations between users and a deployed assistant. You can use this history to improve how your assistants understand and respond to user requests.

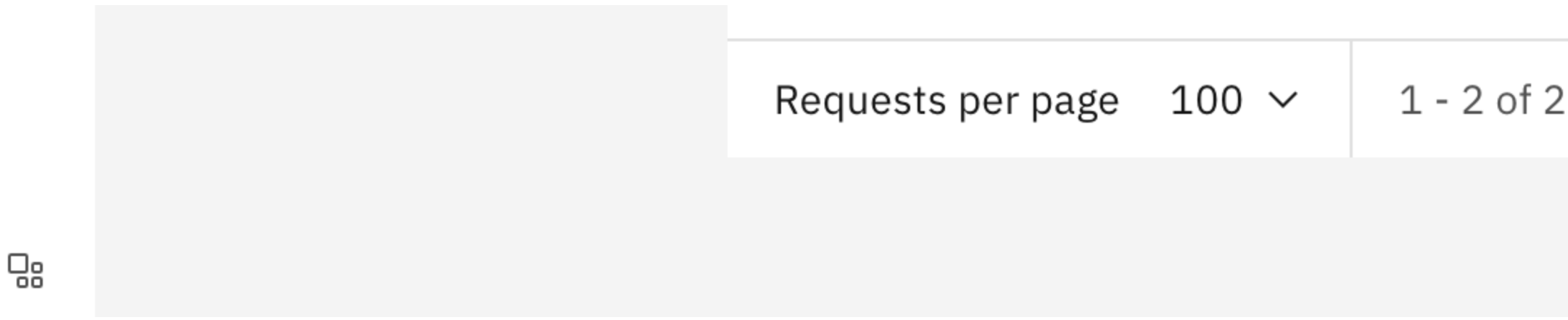
In the **Analyze** page, you can see the summary of the conversations at the topic level in **View by recognition type** tab.



In the **Message log** tab, you see the complete history of conversation between user and the assistant by turns.

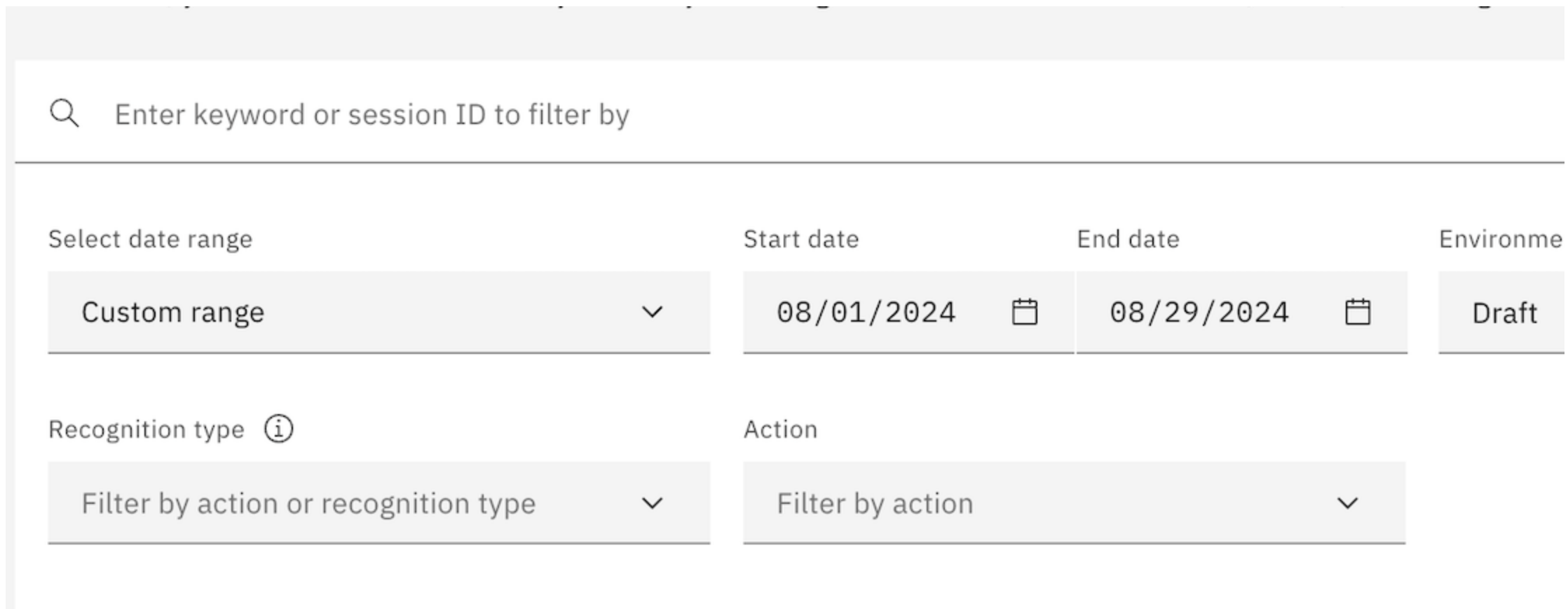
Note: If you have activated dialog, you see the Entities column in the **Message log** tab.





Choosing the environment and time period

To get started, choose the environment and date range you want to analyze. All conversations reflect data based on the environment and the date range you select. When you change the environment or the date range, the conversations on the page update to reflect the new date range. You can also use **Refresh** to ensure the latest data is shown.



Filtering conversations

You can locate specific conversations by filtering the list of conversations. This lets you explore specific areas where your assistant might need improvement or updates to properly handle what your customers are asking about.

You can filter the conversations by:

- **Recognition type**: Select one or more from different type of actions, recognized or unrecognized user requests, search, intents.
- **Action**: Select one or more from user created actions, system created actions or intents.
- **Entity**: Select specific entities. You can choose one or more entities to review.

  The Entities filter appears in the **Message log** tab only if you have activated dialog in your assistant.

Exploring conversations in detail

To explore individual conversations in detail, you can go to one of the following tabs in the **Analyze** page based on the level of details you want to analyze:

- **View by recognition type** Displays the list of conversations with the topic-level details such as request, recognition type, and conversation timestamp and session ID. In the **View by recognition type** tab you see the following columns:
  - The **Requests** column includes the questions or requests the user entered that initiated an action, started a search query, or weren't recognized.
  - The **Recognition type** column shows you the type of action that got triggered from user requests.
  - The **Details** column shows the timestamp of a single conversation and its session ID.

  When you click on a request or timestamp, a panel opens showing the summary of requests, recognition status, and the duration of the conversation.
- **Message log view** Displays the list of conversations between the users and assistant with the details such as consumer input, topic, assigned entity, timestamp, and session ID. In the **Message log** tab you see the following columns:

- The **Customer input** column shows you the customer responses or input.
- The **Recognition type** column shows you the type of action that got triggered from user requests.
- The **Entities** column shows the entities assigned to each customer input. The Entities filter appears in the **Message log** tab only if you have activated dialog in your assistant.
- The **Details** column shows timestamp of a single conversation and its session ID.

When you click on customer input or timestamp, a panel opens showing the conversation between the user and the assistant by turns along with duration and number of recognized input, customer input and searches.

Sending events to Segment

Premium Plus

You can use the Segment extension to send watsonx Assistant events to Segment.

Overview

With this extension, you can use Segment to capture and centralize data about your customers' behavior, including their interactions with your assistant. Events are sent from watsonx Assistant to Segment, making them available to destinations such as data warehouses, raw data tools, and analytic tools.

For more information about the events that are sent to Segment, see Segment event reference.

Adding the extension to the draft environment

To add the Segment extension to your assistant, follow these steps:

1. On the **Integrations** page, scroll to the **Extensions** section and find the tile for the Segment extension.
2. Click **Add**. Review the overview of the extension and click **Add** to configure it for your assistant.

   > ⚠ **Important:** When you first add the Segment extension to an assistant, the configuration settings you provide are applied only to the draft environment. You must complete configuration for the draft environment before you can add the extension in the live environment.

3. In the **Connect** step, click the link to log in to your Segment account in another browser tab.

   If you do not already have a Segment account, click **Sign up for free account** to create one. Verify your email address and complete your profile to activate your account.

4. In the Segment web app, go to your workspace. In the **Sources** section, find the **Add IBM watsonx Assistant Source** tile and click **Add Source**.
5. In the **Source Name** field, type a descriptive name for your watsonx Assistant instance (for example, `Customer Care Assistant`). Click **Create Source**.
6. Click **Copy** to copy the generated key to the clipboard.
7. Go back to the Segment integration settings in the watsonx Assistant interface. In the **Segment key** field, paste the key you copied from the Segment web app in the previous step. Click **Next**.
8. In the **Select events** step, review the list of events watsonx Assistant can send to Segment.

   Each row in the table shows the name of a supported event, along with a brief description.

9. Click the checkboxes to select the events you want to send to Segment. Click **Next**.
10. In the **Review & Confirm** step, review the configuration and click **Finish**.

The Segment extension is now connected to your assistant in the draft environment. Click **Close** to close the integration settings.

Configuring the extension for the live environment

To configure the Segment extension for the live environment, follow these steps:

1. On the **Integrations** page, scroll to the **Extensions** section and find the tile for the Segment extension.
2. Click **Open**. The **Open extension** window opens.
3. In the **Environment** field, select **Live**. Click **Confirm**.
4. Repeat the configuration process, specifying the values you want to use for the live environment.

> **Note:** If you are using multiple environments, follow the same steps to configure the extension for each environment. For more information, see Adding and using multiple environments.

The Segment extension is now available in the environments you have configured, and events will be sent to the destinations configured in your Segment workspace. (For more information about the events that are sent to Segment, see Segment event reference.)

Using watsonx.ai for generative AI capabilities

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Information gathering

You can use watsonx.ai in your assistant to intelligently gather information and address a customer query quickly to avoid repetitive questions. The watsonx.ai uses large language models (LLMs) to recognize the information that is collected from the customer and store each of them in the respective variable for further processing.

Note: This feature is available for Enterprise with data isolation plan.

Note: This feature is available in English, French, German, Spanish, Brazilian Portuguese and Japanese. The watsonx generative AI model that supports English, French, German, Spanish, and Brazilian Portuguese is hosted only in the Dallas and Frankfurt regions. The watsonx generative AI model that supports Japanese is hosted only in the Tokyo regions.

By enabling information gathering, your assistant can analyze a customer's free text response to:

- Gather accurate information
- Complete multiple steps in one step

For more information about collecting information from customer responses, see Collecting information from your customers.

Enabling information gathering capability

To enable the information gathering capability, do the following steps:

1. Go to **Home** > **Actions** > **Global settings** > **Generative AI** > **Information gathering**.
2. Set **Use watsonx.ai information gathering** to `On`.

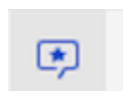
> **Tip:** When you enable intelligent information gathering, settings for existing free text customer responses change to `Skip asking if the answer is mentioned in previous messages`. If you later disable information gathering, revert the settings for free text responses to `Always ask`. For more information, see Skipping steps, always asking steps, or never asking steps.

Adding examples for clarity

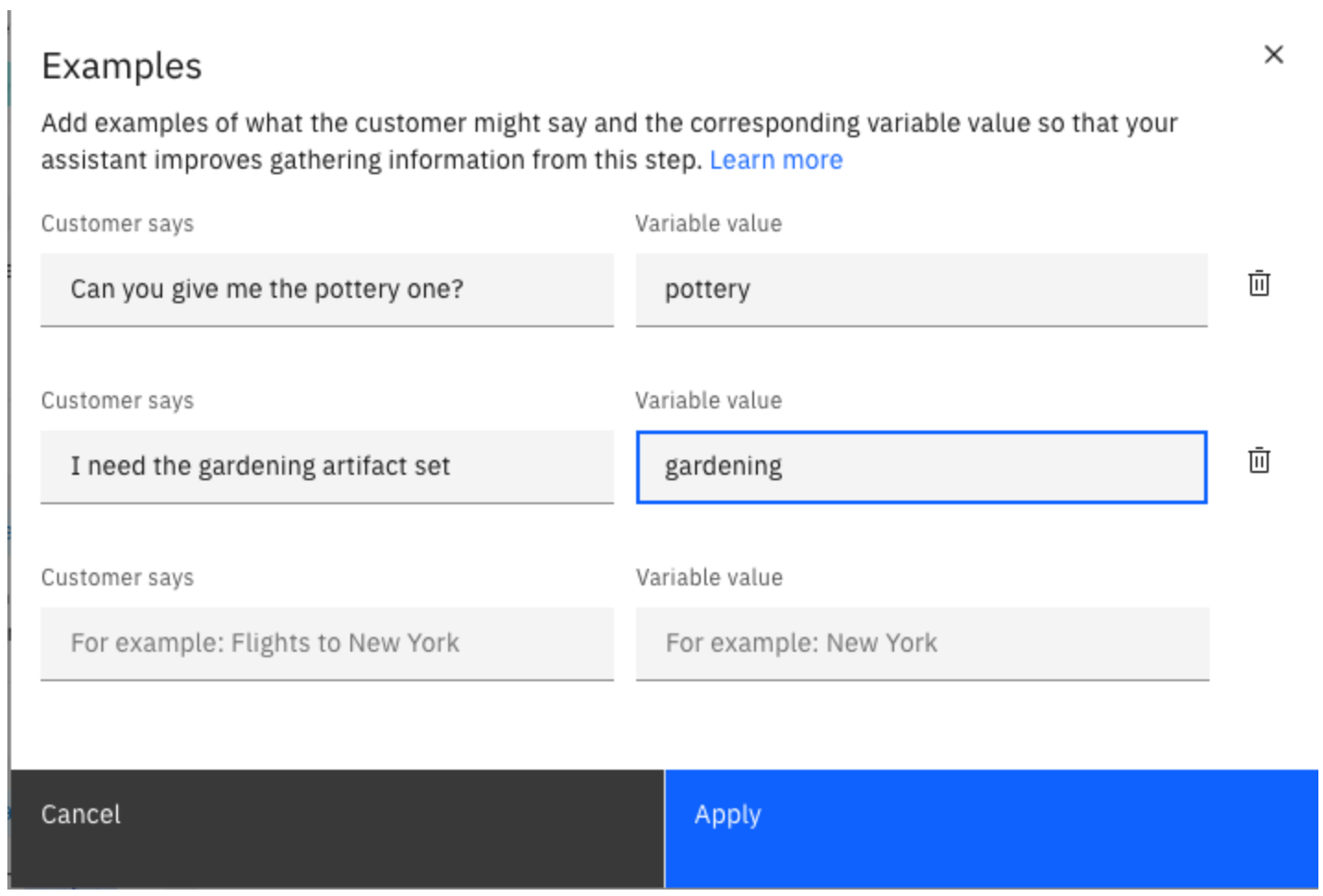
The **Add examples** feature in the action editor improves the capability of your assistant to gather accurate information from the customer response. By using the large language models (LLMs) in *watsonx.ai* and the added examples, your assistant can identify the variables in a customer response and retrieve accurate information.

You can add examples whenever the response from LLMs lacks clarity on a particular topic. The **Add examples** feature is an optional enhancement because watsonx.ai is capable of gathering information accurately even without the examples.

To use the **Add examples** icon, do the following:

1. Go to **Home** > **Actions** > **Editor**.
2. Go to the **Step** where you want to add examples.
3. In **Assistant says**, type the response that you want to display.
4. Click **Define customer response** and select **Free text**.
5. Click **Add examples**  icon, which opens an 'Examples' window.
6. In the 'Examples' window, type any customer relevant topic in `Customer says` and give a variable name in `Variable value`.
7. Click `Apply` to save your changes.

> **Note:** You can add only three examples in the 'Examples' window.



Hate, abuse, and profanity filter

You can use the hate, abuse, and profanity (HAP) filter to maintain an inclusive environment. This feature filters content to provide a positive online atmosphere and a safe community for your customers.

By enabling HAP detection, you can create a more consistent and reliable user experience that safeguards against the rare instances where HAP content might be generated.

Enabling the HAP filter

To enable the HAP filter:

1. Go to **Global settings** > **Generative AI** > **HAP filtering**
2. Click the toggle to turn it on.

The HAP filter applies to the following areas:

- Conversational search
- Content-grounded answering
- General-purpose answering
- AI-guided actions

Consider the following key points when you use the HAP filter:

HAP detection: If HAP is detected, the assistant provides a generic fallback response. For more information, see Conversational search.

Recommendation: When HAP is on, it minimizes the risk of unintended or inappropriate responses.

Conversational search

Plus Enterprise IBM Cloud Pak for Data IBM Software Hub

Important: Starting from **June 1, 2024**, add-on charges are applicable for using the Conversational search feature in addition to your Plus or Enterprise plans. For more information about the pricing plans, see Pricing plans. For more information about terms, see Terms.

Note: This feature is available for Enterprise with data isolation plan.

Use *conversational search* with the search integrations such as IBM Watson® Discovery, Elasticsearch, Custom search, or Milvus to help your assistant extract an answer from the highest-ranked query results and return a text response to the user.

When you enable this feature, search results are provided to an IBM watsonx generative AI model that produces a conversational reply to a user's question.

Important: The watsonx generative AI model is hosted only in the Dallas and Frankfurt regions. By default, assistants in all regions except `Frankfurt` use the model from the `Dallas` region. Assistants in the `Frankfurt` region use the model hosted in the `Frankfurt` region.

Before you begin

You must configure the search integration to enable the conversational search feature. For more information about configuring IBM Watson® Discovery search integration, see Discovery search integration setup. For more information about configuring Elasticsearch integration, see Elasticsearch search integration setup.

Enabling conversational search

You can enable conversational search to give accurate responses to the customer query. In addition, you can enable citations by putting a citation title, which gives the list of references of the source content from where the assistant pulled the responses. You can see the citation title in between the conversational response and the citations.

To enable conversational search, do the following steps:

1. Go to the **Search Integration** window.
2. Set the **Conversational search** toggle to `On`.
3. Choose the type of conversational search based on the context using contextual awareness.
    - Single-turn conversational search

    For contexts that require only current input to retrieve search results and to generate answers, choose **Single-turn**.
    - Conversational search using entire conversation

    For context-dependent questions, which might consider previous inputs, choose **Entire conversation**.

    Note: Entire conversation uses the whole session to continue the conversation. It might bring back subjects that are no longer in the scope of

the conversation.

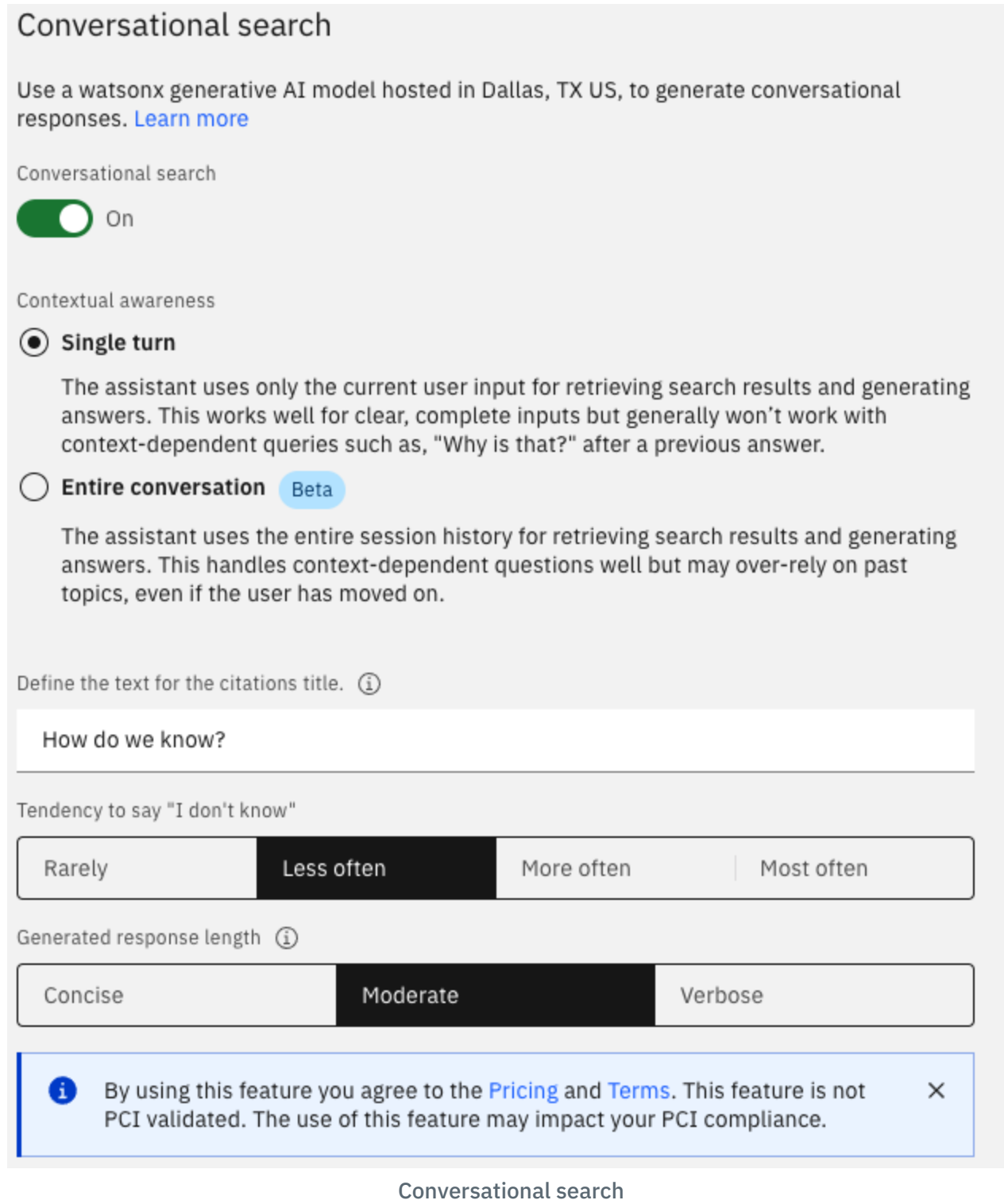
4. In the **Define the text for the citation title**, type `How do we know?`.

> **Tip:** The **Define the text for the citation title** is enabled only when **Conversational search** toggle is switched to `On`.

> **Note:** The web chat integration does not support the citation title feature.

5. In the **Tendency to say I don't know**, select the tendency to use. By default, `Less often` is selected.

6. Click **Save**.



Tuning conversational search's tendency to say "I don't know"

You can tune the tendency of your assistant to say "I don't know" in conversational search by selecting one of the following options in the **Tendency to say I don't know** section:

- `Rarely`
- `Less often`
- `More often`
- `Most often`

When your assistant generates a conversational search response, it evaluates the response and calculates a response confidence score. The assistant compares the response confidence score against your selection in the **Tendency to say I don't know** section. If the response confidence score is comparatively high, the assistant responds to the user query with the generated response. If the response confidence score is comparatively low, the assistant does one of the following actions:

- Responds with an "I don't know" message
- Falls back to the "No matches" action per the Search routing configuration in your assistant

Table 1. Tendency to say “I don’t know” options shows the options available in the **Tendency to say “I don’t know”** section.

Tendency to say “I don’t know”	Response confidence threshold	Assistant behavior
Rarely	Lowest	The assistant rarely says “I don’t know” because it compares the response confidence score against the lowest threshold. Therefore, your assistant gives a generated response to the user almost all the time. However, the assistant most likely presents inaccurate or irrelevant responses to the user.
Less often	Lower	The assistant says “I don’t know” more often than the Rarely option.
More often	Higher	The assistant says “I don’t know” more often than the Less often option.
Most often	Highest	The assistant says “I don’t know” more often than the More often option because it compares the response confidence score against the highest threshold. However, the assistant most likely presents accurate or relevant responses to the user. In addition, the assistant presents fewer generated responses to the users and more often responds with an “I don’t know” message or falls back to the “No matches” action.

Tuning the generated response length in conversational search

The generated response-length feature in IBM Watson Assistant customizes response lengths to best meet your needs.

You can choose from three response lengths: concise, moderate, and verbose. This feature adjusts the length of the responses that your assistant gives to better fit your needs in conversational search. The default setting is moderate, but you can change it as needed:

Response length	Description
Concise	Responses are shorter and to the point, which is ideal for straightforward queries.
Moderate	Responses balance detail and conciseness, making them suitable for most general inquiries.
Verbose	Responses provide more detailed and comprehensive information that is suitable for complex queries or when a thorough explanation is needed.

Note: The response-length feature affects the average length of responses that watsonx Assistant generates. Although it aims to match the specified length, actual responses vary because of the complexity of user input and the inherent limitations of the large language model (LLM).

Configuring your assistant to use the conversational search

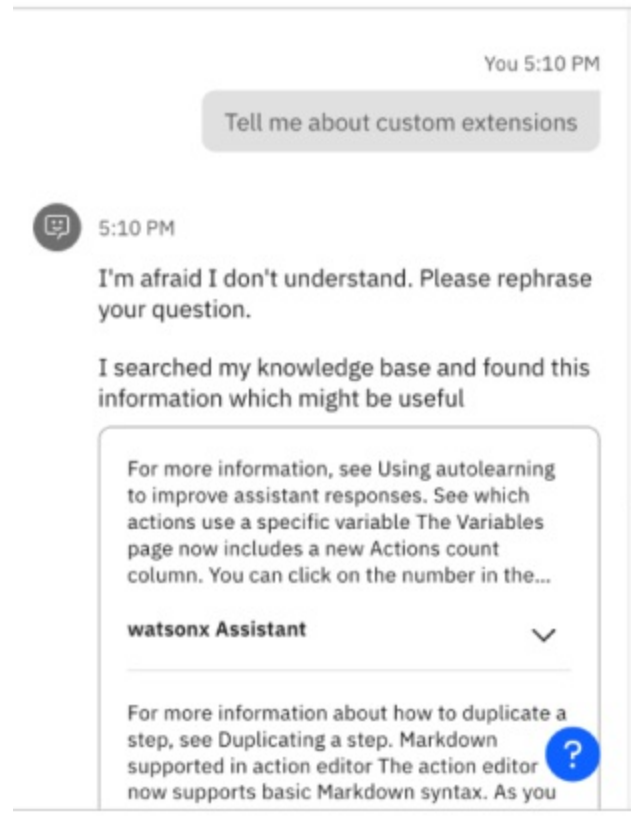
After you enable **Conversational search**, you must configure the **Search routing** setting to route your assistant responses to conversational search when no action matches the user response. For more information about the **Search routing** configuration, see the Configuring the search routing when no action matches topic. To configure your assistant to route to conversational search for specific topics or actions, you can add search as a step in a new or existing action.

When your assistant receives no search results from Elasticsearch or Discovery in response to a user query or when its connection to Elasticsearch or Discovery fails, your assistant responds to the user with a failure message. You can configure the failure messages for no search results and a failed connection in the **Search integration** settings.

Testing conversational search

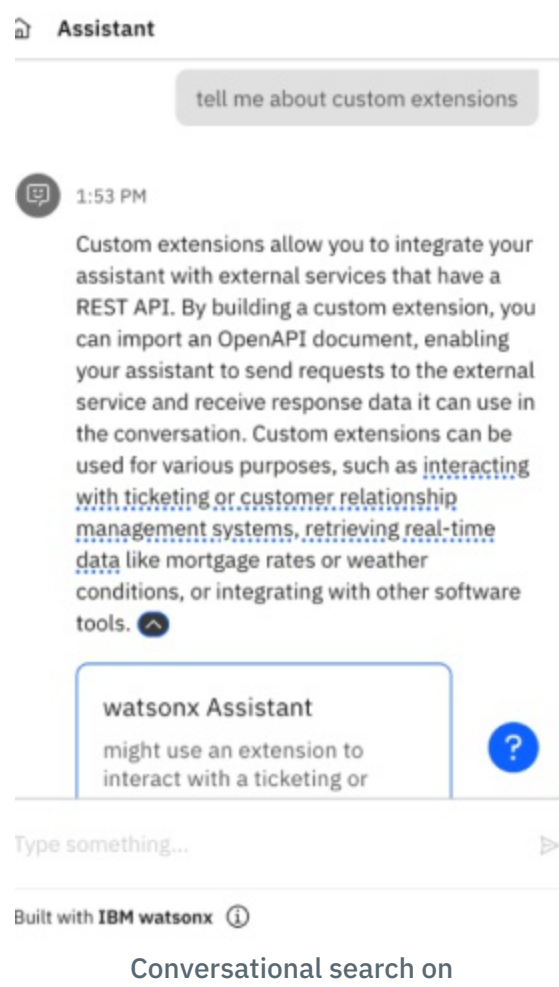
You can test the conversational search in actions preview, the preview page, or by using the preview link.

In this example, the user asks, Tell me about a custom extension. Search results are pulled from your knowledge base when the conversational search is Off. In this case, the answer is returned as a list of cards that are relevant to custom extensions.



Conversational search off

When the conversational search is `On`, the same search results are pulled from your knowledge base. The results are passed to an IBM watsonx generative AI model. This model produces a conversational reply to the user's question, in the form of a text response about custom extensions.



Conversational search on

Debugging the failures in conversational search

If your calls to the conversational search fail, you might want to debug the problem by seeing the detailed information about what is being sent to and returned from the system API.

For more information, see Debugging failures for conversational search

Streaming response for conversational search

Streaming response for the conversational search uses watsonx.ai capabilities to provide continuous, real-time responses in your assistant. By default, the streaming response is disabled for the web chat and the assistant preview panels.

By using the streaming response support feature, you can reduce the wait time for the response.

To enable streaming response:

1. Go to **Home** > **Preview** > **Customize web chat**.
2. Click the **Styles** tab.

3. Set the **Streaming** toggle to `On`.
4. Click **Save and exit**.

Adding more assistants

Note: If you're looking for the AI Assistant Builder documentation, see Building AI Assistants.

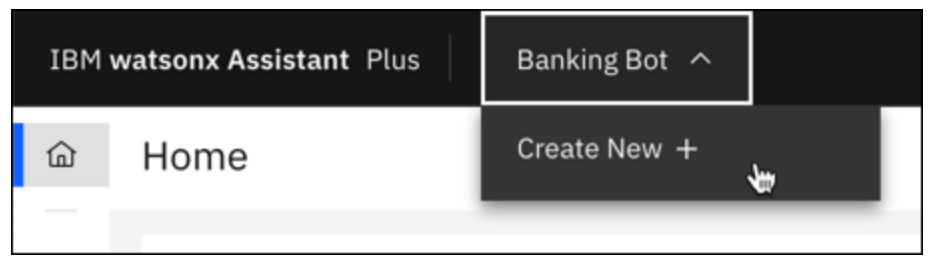
You can create multiple assistants in your instance. The number of assistants you can create depends on your plan.

Plan	Assistants per instance
Enterprise	30
Plus	10
Trial	10
Lite	3

Service plan details

If you need to add more assistants, follow these steps:

1. In the navigation, click the name of your current assistant, and then choose **Create New**.



Create new assistant

2. Add details about the new assistant.
   - **Assistant name** (Required): Enter a name no more than 100 characters in length.
   - **Description** (Optional): Enter a description no more than 200 characters in length.
   - **Assistant language**: Select a language for the assistant to use in conversations. For more information, see language support.
3. Click **Create assistant**.

If you're using the classic experience, follow these steps to add an assistant:

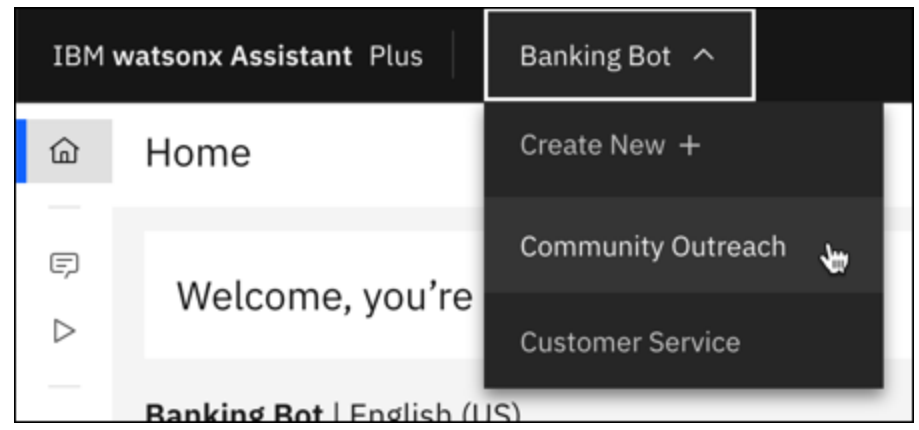
1. Click the **Assistants** icon to open the Assistants page.
2. Click **Create assistant**.
3. Add details about the new assistant:
   - **Name**: A name no more than 100 characters in length. A name is required.
   - **Description**: An optional description no more than 200 characters in length.
4. Click **Create assistant**.
5. Add a skill to the assistant.

   Note: You can choose to add an existing skill or create a new one.

Switching between assistants

To switch to another assistant in your instance:

1. In the navigation, click the name of your current assistant.



Switch assistants

2. Choose the assistant that you want to open.

If you're using the classic experience, follow these steps to switch assistants:

1. Click the **Assistants** icon to open the Assistants page.
2. Choose an assistant you want to open.

Renaming or deleting assistants

If necessary, you can rename or delete an assistant. You can't change the language of an assistant.

Renaming an assistant

You can change the name and description of an assistant after you create it.

To rename an assistant, follow these steps:

1. Switch to the assistant you want to rename.
2. Open **Assistant settings**.
3. In **Details**, enter a new name or description.
4. Click **Save**.

If you're using the classic experience, follow these steps to rename an assistant:

1. From the Assistants page, find the assistant that you want to rename.
2. Click the ⋮ icon, and then choose **Rename**.
3. Edit the name, and then click **Save**.

Deleting an assistant

To delete an assistant, follow these steps:

1. Switch to the assistant you want to delete.
2. Open **Assistant settings**.
3. Click the **Delete assistant** button on the settings page. A confirmation question appears.
4. To confirm, type the word `DELETE`, then click **Delete**.

If you're using the classic experience, follow these steps to delete an assistant:

1. From the Assistants page, find the assistant that you want to delete.
2. Click the ⋮ icon, and then choose **Delete**. Confirm the deletion.

Administering your instance

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Managing access

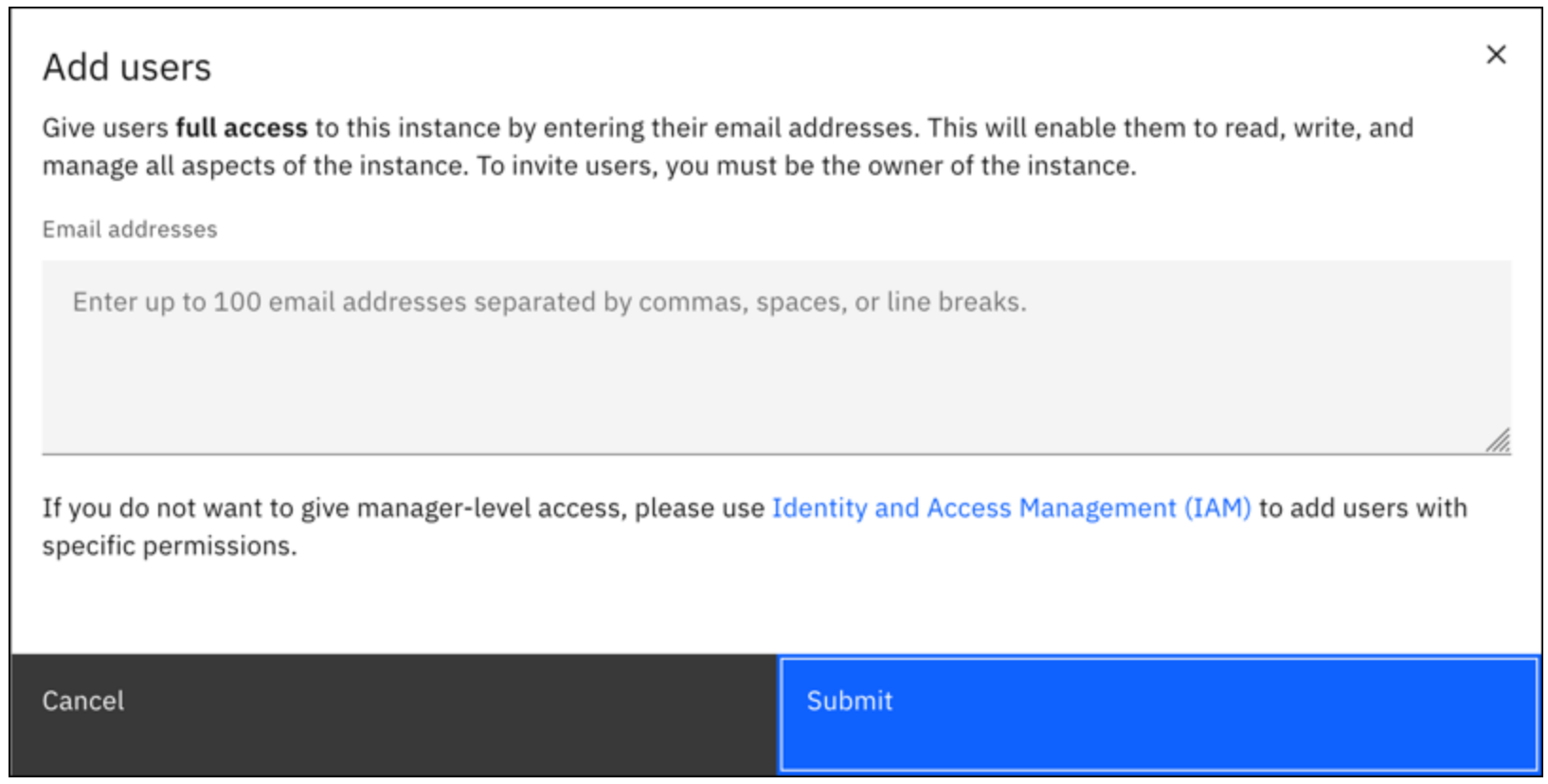
If you need to collaborate with others on your assistants, you can quickly add users to your service instance from the **Manage** menu. Or to tailor specific access to your assistants, use the Identity and Access Management (IAM) page in IBM Cloud.

Adding users from the Manage menu

In watsonx Assistant, each assistant contains all the draft and live resolution methods (actions and search integration) and channels you add (such as web chat, Facebook, or Slack). The simplest way to provide access is to add users to your watsonx Assistant service instance with manager access to all assistants. Users get all the privileges that they need to build and deploy any assistant.

To quickly add users with manager access to all assistants, complete the following steps:

1. Open the **Manage** menu .
2. Click **Add users**.
3. Enter the email addresses of the users that you want to provide full access to. Separate email addresses with commas, spaces, or line breaks.



Add users

Important: Adding users from this menu enables them to read, write, and manage all assistants in the service instance.

4. Click **Submit**.

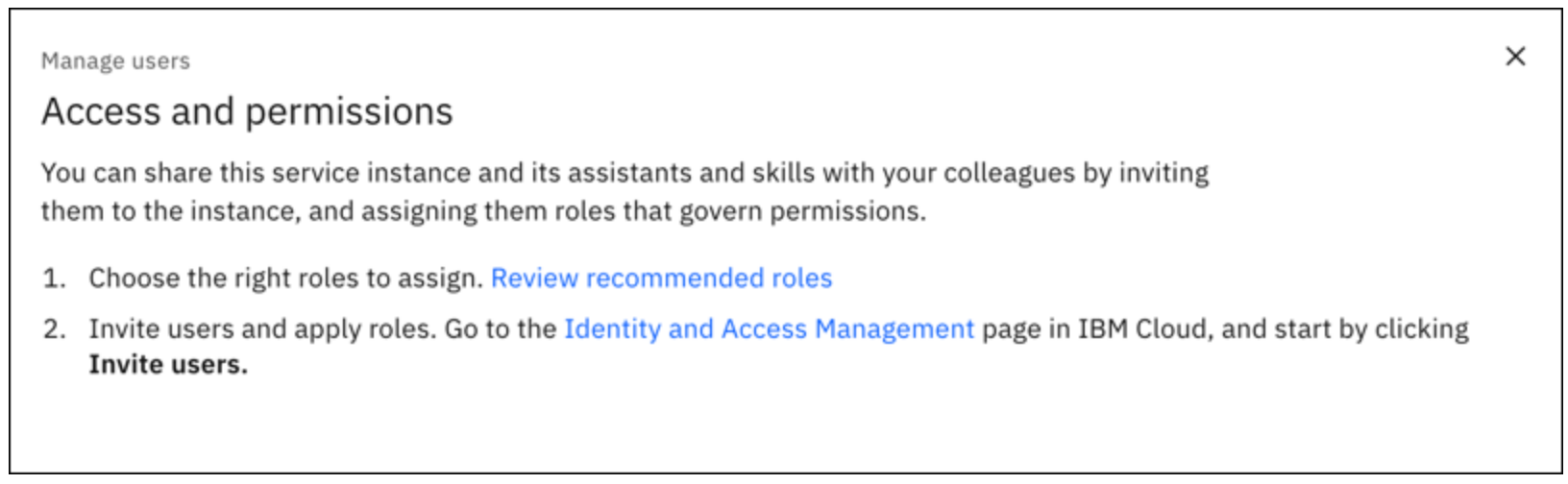
After you click **Submit**, any user that you invite receives an email to access the instance. After they accept the invite, they can open the service instance and manage all assistants.

Managing access with Identity and Access Management

Another way to add users to your assistants is using Identity and Access Management (IAM). If you want to add users, and you don't want them to have full Manager access, use IAM to add them. From IAM, you can also manage access roles of those users that are already added to your assistants.

Opening Identity and Access Management

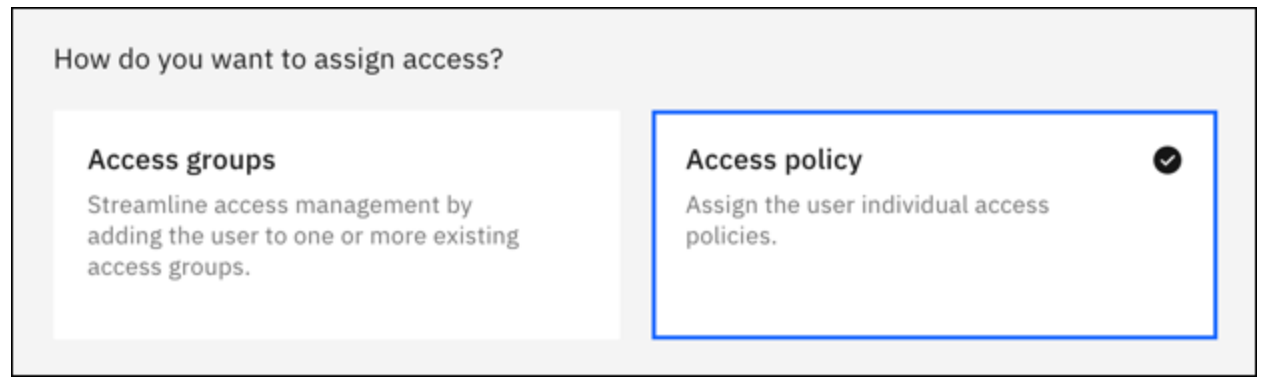
1. Open the **Manage** menu .
2. Click **Manage users**.
3. In **Access and permissions**, click **Identity and Access Management** in step 2.



Access and permissions

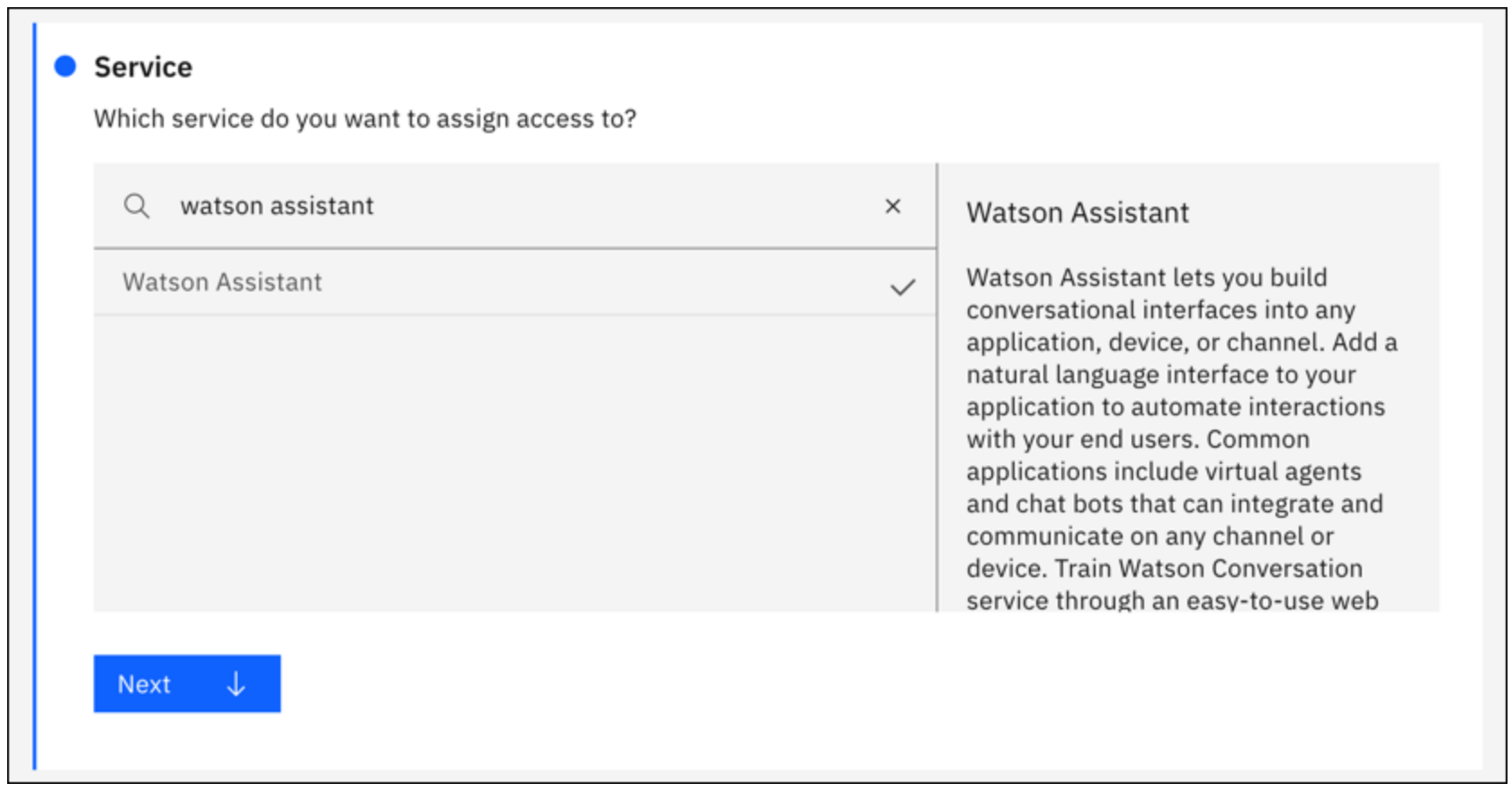
Adding users in Identity and Access Management

1. In IAM, click **Invite users**.
2. Enter the email address of the person who needs access.
3. In **How do you want to assign access?** , choose **Access policy**.



Access policy

4. In **Service**, choose **watsonx Assistant**, then click **Next**.



Service

5. In **Resources**, choose either **All resources** or **Specific resources**.

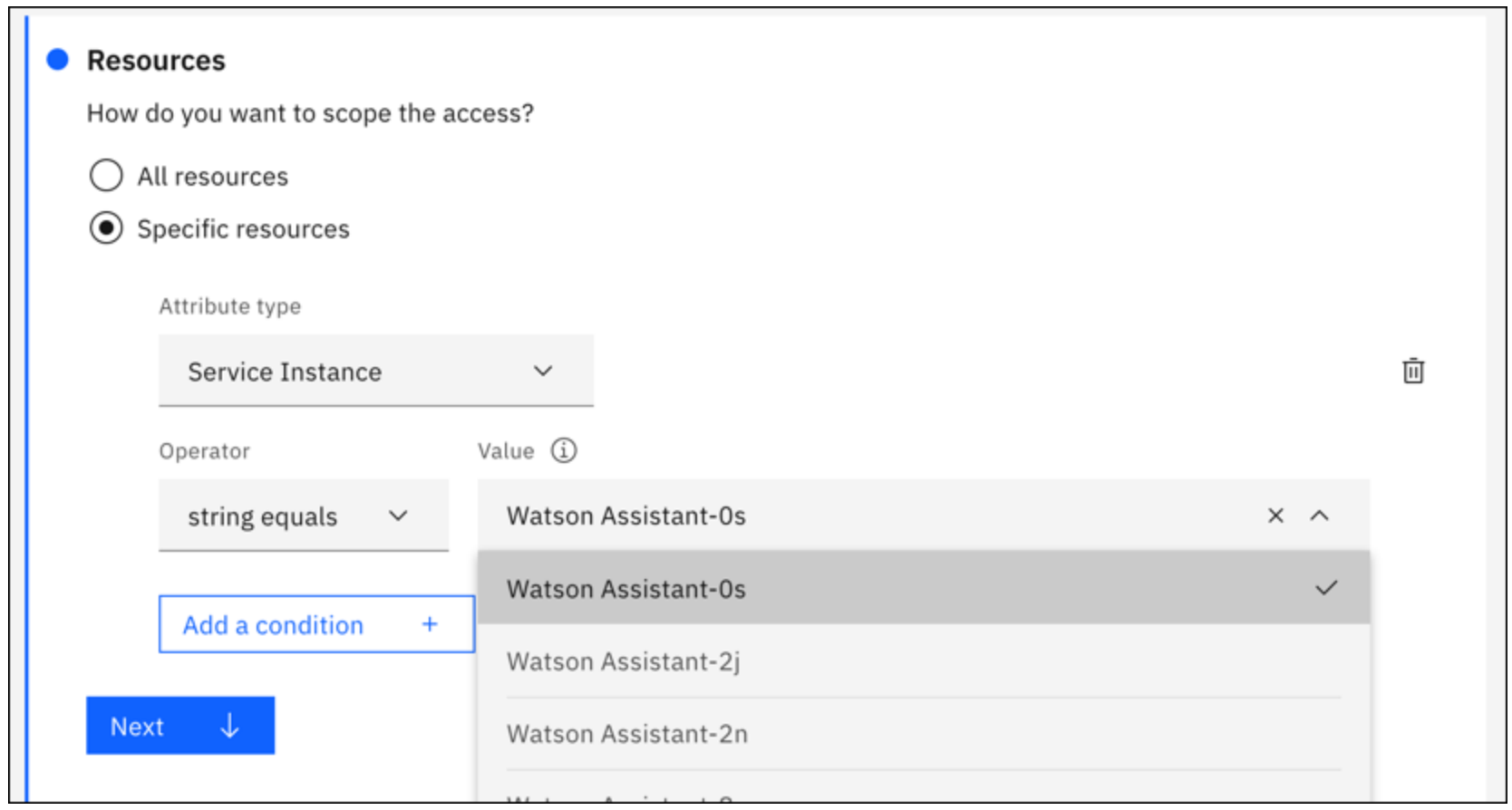
   If you choose **All resources**, the user can access all the instances of watsonx Assistant in your account.

   If you choose **Specific resources**, you can narrow access in **Attribute type**. With this setting, you might need to add multiple access policies for a user to grant the correct access. For more information, see Example of limiting access to one assistant.

   Choices include:

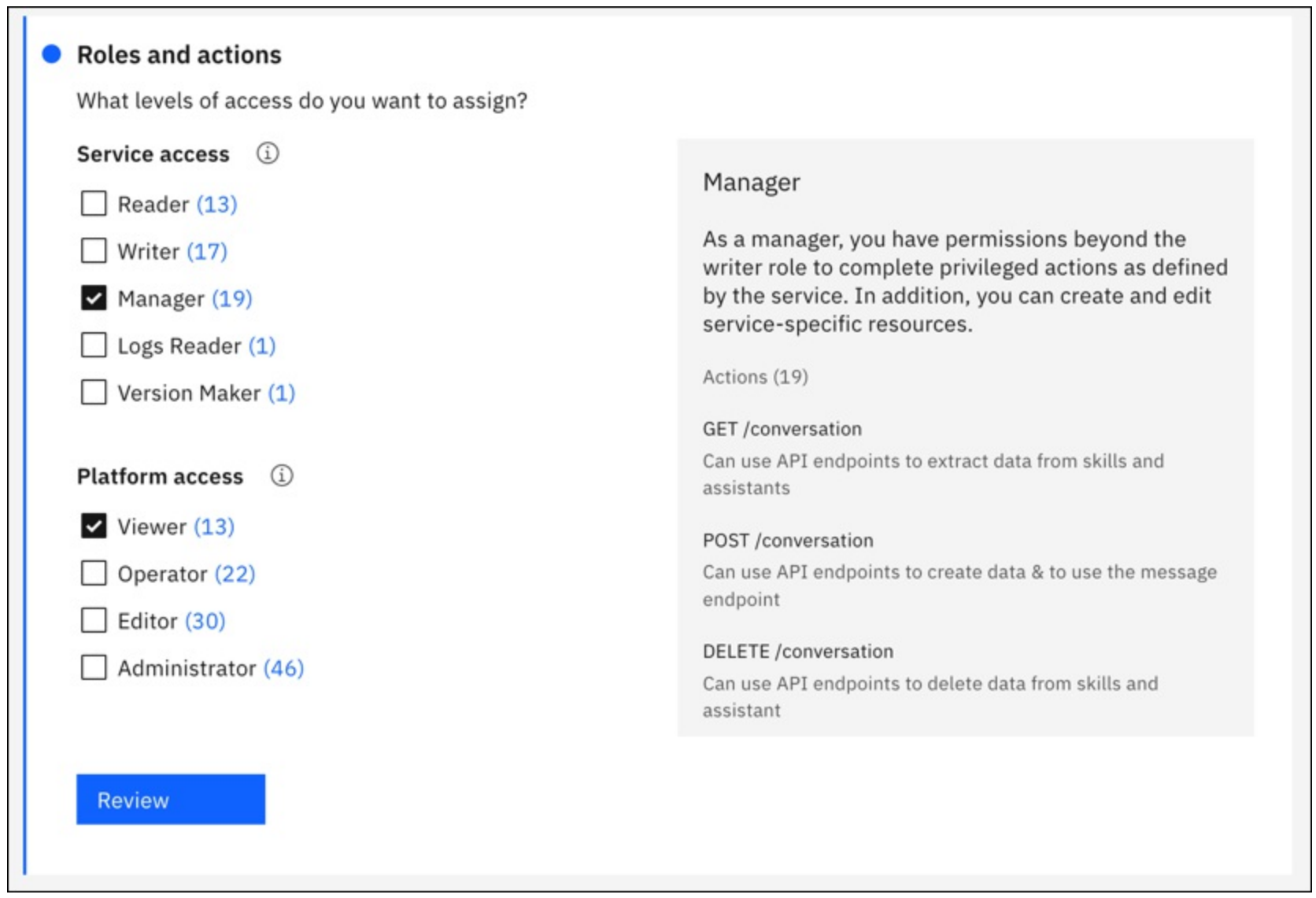
Resource attribute type	Description
Service Instance	Choose a specific service instance of watsonx Assistant
Assistant, Environment, or Skill ID	Enter the ID value for the resource. Use the **Assistant IDs and API details** section in **Assistant settings** to get the ID values for your assistant, environments, action skill, or dialog skill.
Resource Type	If you enter an ID value, choose Assistant ID, Environment ID, or Skill ID to identify the ID type
Region	Choose a specific region (for example, Dallas or London)
Resource Group	Enter or choose a resource group that you created

Attribute types



Resources

6. In **Roles and actions**, select the service role that you want the user to have. Service access controls what a person can do in watsonx Assistant. Next, select the platform role that you want the user to have. Platform access controls a person's ability to access a service instance in IBM Cloud. Then, click **Review**.

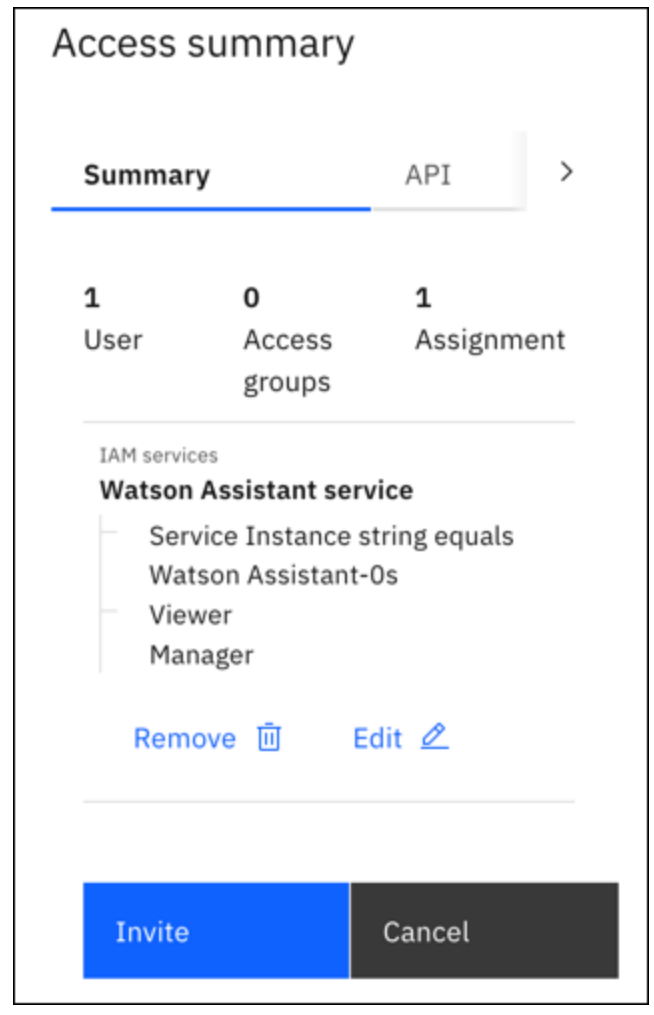


Roles and actions

7. Click **Add** to add the access policy.

Platform and service access

8. To finish, click **Invite**.



Invite

The user that you invited appears in your list with the status of **Processing**. After they accept the invite, status changes to **Active**, and the user can work on your assistant with you.

Platform roles

A platform role controls a user's ability to open and work with a service instance in IBM Cloud.

Important: At a minimum, each user needs the *Viewer* platform role for a service instance

Role	Open	Modify	Delete	Manage access
Viewer	✓			
Operator	✓	✓		
Editor	✓	✓		
Administrator	✓	✓	✓	✓

Platform role details

Service roles

A service role controls what a person can do within each service instance.

Role	Description
Reader	Read-only access to a resource. Use with Logs Reader to provide access to Analytics.
Writer	Create and edit within a resource.
Manager	Manage everything in a resource.
Logs Reader	Use Logs Reader in combination with the Reader or Writer role to provide access to Analytics.
Version Maker	Create or delete versions of an assistant. Doesn't provide publish access.

Service role details

This table explains the minimum service roles that are required for common tasks in an assistant.

Task	Resource	Minimum service role required
Assistant		
Create assistant	Service instance	Writer
View assistant settings	Assistant	Writer
View assistant ID	Assistant	Writer
Update assistant settings	Assistant	Writer
Enable or disable dialog	Assistant	Writer
Delete assistant	Service instance	Writer
View assistant list	Assistant	Reader
Actions		
Create action	Action skill	Writer
Update action	Action skill	Writer
Delete action	Action skill	Writer
Download actions JSON file	Action skill	Reader
Upload actions JSON file	Action skill	Writer
Copy action	Action skill (in destination assistant)	Writer
Create collections	Action skill	Writer
Update collections	Action skill	Writer
Delete collections	Action skill	Writer

Read collections	Action skill	Reader
Publish		
Publish version	Environment	Writer
Create version without publishing	Assistant	Writer or Version Maker
Delete unpublished version	Assistant	Writer or Version Maker
Download version	Assistant	Reader
Environments		
Create environment (Enterprise plan only)	Assistant	Writer
Update environment settings	Environment	Writer
Delete environment (Enterprise plan only)	Assistant	Writer
Integrations		
Add integration	Service instance	Writer
Update integration	Service instance	Writer
Delete integration	Service instance	Writer
After Integrations		
Create integration	Assistant, Service instance	Writer
Update integration	Assistant, Service instance	Writer
Delete integration	Assistant, Service instance	Writer
Dialog		
Create intent	Dialog skill	Writer
Update intent	Dialog skill	Writer
Delete intent	Dialog skill	Writer
Import intents	Dialog skill	Writer
Export intents	Dialog skill	Reader
View intents	Dialog skill	Reader
Create entity	Dialog skill	Writer
Update entity	Dialog skill	Writer
Delete entity	Dialog skill	Writer

Import entity	Dialog skill	Writer
Export entity	Dialog skill	Reader
View entity	Dialog skill	Reader
Download intents and entities	Dialog skill	Reader

Minimum service role details

Example of limiting access to one assistant

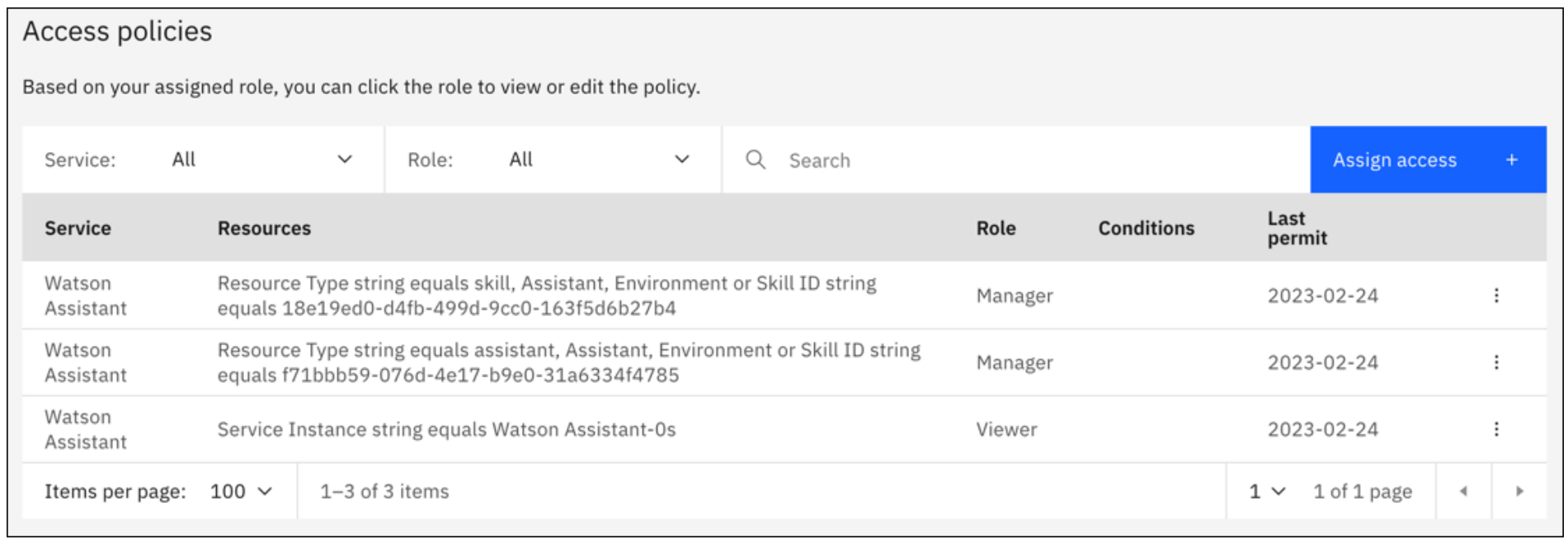
This example explains how to follow the steps in Adding users from the Manage menu and set specific resources that limit a user to building and publishing actions in one assistant. For each user, you need to add three access policies that identify the service instance, assistant ID, and skill ID. (Use **Assistant settings** to get the ID values for your assistant and action skill.)

This table lists the values that you need to add for each policy:

Policy	Specific resources	Value	Service role	Platform role
1	Service Instance	Choose the instance that includes the assistant	None	Viewer
2	Resource Type	Assistant ID	Manager	None
2	Assistant, Environment, or Skill ID	ID for the assistant	Manager	None
3	Resource Type	Skill ID	Manager	None
3	Assistant, Environment, or Skill ID	ID for the action skill	Manager	None

Settings to limit access to one assistant

The access policies for your user should look like this example:



Access policies example

With this set of access policies, your user can build and publish actions in one assistant. The user can't add integrations because the service instance is set to `Reader`. The user has read-only access to other assistants but can't build or publish actions.

Managing your plan

Learn about:

- Plan information
- Upgrading your plan

Plan information

Billing for the use of watsonx Assistant is managed through your IBM Cloud® account.

The metrics that are used for billing purposes differ based on your plan type. You can be billed based on the number of API calls made to a service instance or on the number of active users who interact with the instance.

For answers to common questions about subscriptions, see How you're charged.

Explore the watsonx Assistant service plan options.

Paid plan features

The following features are available only to users of a Plus plan or higher. Plus

- Phone integration
- Private endpoints
- Search
- watsonx Assistant v2 API
- Log webhook

Note: The v2 Logs API is available with a free trial of the Plus plan.

The following features are available only to users of Enterprise plans. Enterprise

- Activity tracking
- Multiple environments
- Override system defaults for response modes
- Sending events to Segment

The plan type of the service instance you are currently using is displayed in the page header. You can upgrade from one plan type to another. For more information, see Upgrading.

User-based plans explained

Unlike API-based plans, which measure usage by the number of API calls made during a month, the Plus and Enterprise plans measure usage by the number of monthly active users.

A monthly active user (MAU) is any unique user who has at least one interaction with your assistant or custom application over the calendar billing month.

A unique user is recognized by the user ID that is associated with the person that interacts with your assistant. The web chat and other built-in integrations set this property for you automatically.

You can calculate MAUs on your own, for both IBM Cloud and IBM Cloud Pak for Data. To calculate MAUs, use the logs endpoint to export conversations. For a particular month, count the number of unique user IDs found in the results. User IDs with more than 50 messages (API calls) in a month are counted more than once for every 50 messages. In a common use case, where each user ID represents a customer conversing with an assistant, the average number of messages per user are typically much fewer than 50 messages, so it's unusual to count a user ID more than once.

Specifying the user ID with the REST API

If you are using a custom client with the watsonx Assistant API, you must set the `user_id` property in the message payload your client sends to the `message` method. The `user_id` property is specified at the root of the request body, as in this example:

```
$ {
  "input": {
    "message_type": "text",
    "text": "I want to cancel my order"
  },
  "user_id": "my_user_id"
}
```

Important: In some older SDK versions, the `user_id` property is not supported as a top-level method parameter. As an alternative, you can specify `user_id` within the nested `context.global.system` object.

For more information about the `user_id` property, see the API reference documentation:

- v2 stateless `/message`
- v2 stateful `/message`

If the user ID is not specified

If you are using a custom client application and do not set a `user_id` value, the service automatically sets it to one of the following values:

- **session_id (v2 API only)**: A property defined in the v2 API that identifies a single conversation between a user and the assistant. A session ID is provided in `/message` API calls that are generated by the built-in integrations. The session ends when a user closes the chat window or after the inactivity time limit is reached.

  > **Note:** If you use the stateless v2 message API, you must specify the session_id with each message in an ongoing conversation (in `context.global.session_id`).

- **conversation_id (v1 API only)**: A property defined in the v1 API that is stored in the context object of a `/message` API call. This property can be used to identify multiple `/message` API calls that are associated with a single conversational exchange with one user. However, the same ID is only used if you explicitly retain the ID and pass it back with each request that is made as part of the same conversation. Otherwise, a new ID is generated for each new `/message` API call.

If the same person chats with your assistant on three separate occasions over the same billing period, how you represent that user in the API call impacts how the interactions are billed. If you identify the user interaction with a `user_id`, it counts as one use. If you identify the user interaction with a `session_id`, then it counts as three uses because a separate session is created for each interaction.

Design any custom applications to capture a unique `user_id` or `session_id` and pass the information to watsonx Assistant. Choose a non-human-identifiable ID that doesn't change throughout the customer lifecycle. For example, don't use a person's email address as the user ID. In fact, the `user_id` syntax must meet the requirements for header fields as defined in RFC 7230.

The built-in integrations derive the user ID in the following ways:

- For Facebook integrations, the `user_id` property is set to the sender ID that Facebook provides in its payload.
- For Slack integrations, the `user_id` property is a concatenation of the team ID, such as `T09LVDR7Y`, and the member ID of the user, such has `W4F8K9JNF`. For example, `T09LVDR7YW4F8K9JNF`.
- For web chat, you can set the value of the `user_id` property.

Billing is managed per monthly active user per service instance. If a single user interacts with assistants that are hosted by different service instances that belong to the same plan, each interaction is treated as a separate use. You are billed for the user's interaction with each service instance separately.

Handling anonymous users

If your custom application or assistant interacts with users who are anonymous, you can generate a randomized universally unique ID to represent each anonymous user. For more information about UUIDs, see RFC 4122.

- For web chat, if you do not pass an identifier for the user when the session begins, the web chat creates one for you. It creates a first-party cookie with a generated anonymous ID. The cookie remains active for 45 days. If the same user returns to your site later in the month and chats with your assistant again, the web chat integration recognizes the user. And you are charged only once when the same anonymous user interacts with your assistant multiple times in a single month.

If an anonymous user logs in and later is identified as being the same person who submitted a request with a known ID, you are charged twice. Each message with a unique user ID is charged as an independent active user. To avoid this situation, you can prompt users to log in before you initiate a chat. Or, you can use the anonymous user ID to represent the user consistently.

Data centers

IBM Cloud has a network of global data centers that provide performance benefits to its cloud services. See IBM Cloud global data centers for more details.

You can create watsonx Assistant service instances that are hosted in the following data center locations:

Location	Location code	API location
Dallas	us-south	N/A
Frankfurt	eu-de	fra
Sydney	au-syd	syd
Tokyo	jp-tok	tok

London	eu-gb	lon
Washington DC	us-east	wdc

Data center locations

Upgrading your plan

You can explore the watsonx Assistant service plan options to decide which plan is best for you.

The page header shows the plan that you are using today. To upgrade your plan, complete these steps:

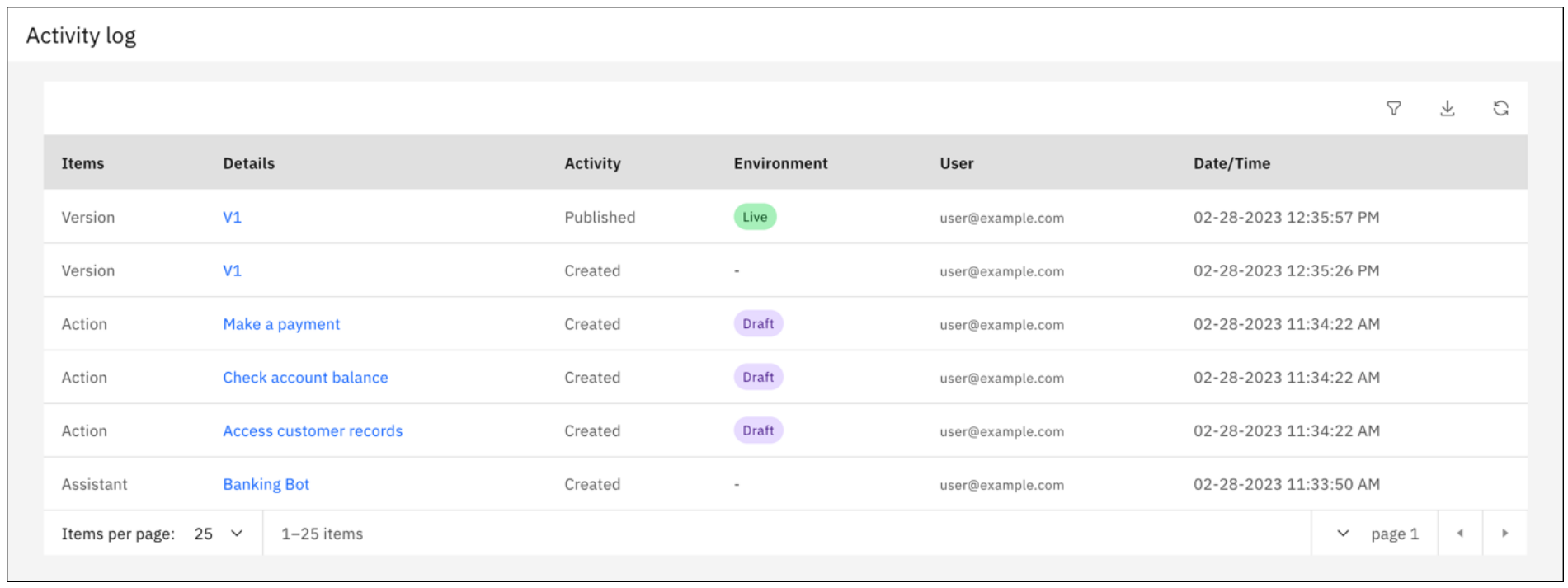
1. Do one of the following things:
   - **Trial plan only**: The number of days that are left in your trial is displayed in the page header. To upgrade your plan, click **Upgrade** from the page header before your trial period ends.
   - For all other plan types, click **Manage**, and then choose **Upgrade** from the menu.
2. From here, you can see other available plan options. For most plan types, you can step through the upgrade process yourself.
   - If you upgrade to an Enterprise with Data Isolation plan, you cannot do an in-place upgrade of your service instance. An Enterprise with Data Isolation plan instance must be provisioned for you first.
   - You cannot change from a Trial plan to a Lite plan.

For answers to common questions about subscriptions, see the How you're charged.

Activity log

IBM Cloud | Plus | Enterprise | IBM Cloud Pak for Data | IBM Software Hub

Use the *activity log* to track changes. It gives you visibility into the modifications that are made to your assistant. It is available for Plus plans and higher.



Activity log

Available days of activity depend on your plan.

Plan	Days of activity
Plus	30
Enterprise	90
Premium	90

Plan information

User access to the activity log

To access and view the log, users need at least the *Reader* service role for an assistant. Users might not be able to open links for specific events, depending on their access to all the items in an assistant.

For more information, see Managing access.

Using the activity log

To open the log, click the **Activity log** icon.

The activity log includes these columns of data:

Column	Description
Item	Describes what was impacted by the change that was made within the assistant. For more information, see Items and activity tracked.
Details	Shows details about the item that was changed. For example, for an action, this column lists the specific action name and provides a link to open it.
Activity	Describes the change that was made. For more information, see Items and activity tracked.
Environment	The environment in which the activity took place or the end destination of an activity.
User	Email address of the user that made the change.
Date/Time	Date and time of the activity in the format DD-MM-YYYY hr:min:sec AM/PM

Activity log columns

Filtering

Click the **Filter** icon to filter the log. You can filter on:

- Item
- Details
- Activity
- Environment
- Environment ID
- User (requires exact match)
- Date range

Sorting

To sort the log, click either the **Details** or **Date/Time** columns.

Downloading

You can download the activity log as a CSV file. You can either download all activity, or filter the activity log by any of filtering choices.

To download the activity log, click the **Download** icon.

Items and activity tracked

For the early access release, here's the current list of items and activity that are tracked. Events are tracked if they complete successfully.

Item	Activity	Notes
Assistant	Created, Dialog activated, Dialog deactivated	
Assistant Settings	Updated	

Action	Created, Updated, Deleted	• Each save (auto or manual) is tracked as an update. • Changes to action settings are tracked as an update.
Variable	Created, Updated, Deleted	
Saved Response Type	Created, Updated, Deleted	
Version	Created, Published, Deleted	
Environment	Created, Updated, Deleted	
Intent	Created, Updated, Deleted	• Each save (auto or manual) is tracked as an update. • A renamed intent retains the old name in previous entries and won't have a link.
Entity	Created, Updated, Deleted	• Each save (auto or manual) is tracked as an update. • A renamed entity retains the old name in previous entries and won't have a link.
Dialog Node	Created, Updated, Deleted	Each save (auto or manual) is tracked as an update.
Search	Updated	
Channel Integrations	Created, Updated, Deleted	• All three activities are applicable for the integrations of Phone, Facebook Messenger, Slack, SMS, WhatsApp with Twilio, Microsoft Teams, and Genesys Bot Connector. • For the Web Chat integration, only the Updated activity is applicable.
Custom Extensions	Created, Deleted, Added, Updated, Removed	All the activities are applicable for Custom Extension Name.
Segment Extension	Added, Updated, Removed	
Dialog—Entity Value Name	Created, Updated, Deleted	
Dialog—Entity Synonym Name	Created, Updated, Deleted	
Dialog—Intent Example Name	Created, Updated, Deleted	
Collections	Created, Updated, Deleted	All the activities are applicable for Name of that collection.

Items and activity

Note: If an item was triggered by the API, the activity log shows the account owner of the API key, even if the key is shared with another person who is triggered the event.

What isn't included in the log

These items aren't included in the activity log:

- Copying an action from another assistant
- Uploading actions, intents, entities aren't included as created
- Changing actions global settings
- Reverting a version to draft
- Adding or deleting entity values and synonyms
- Creating, updating, or deleting channel integrations
- Creating, updating, or deleting custom extensions

Activity tracking events watsonx Assistant

IBM Cloud | Enterprise | Premium

IBM Cloud services, such as watsonx Assistant, generate activity tracking events. This applies to both the classic and new experiences of watsonx Assistant.

Activity tracking events report on activities that change the state of a service in IBM Cloud. You can use the events to investigate abnormal activity and critical actions and to comply with regulatory audit requirements.

You can use IBM Cloud Activity Tracker Event Routing, a platform service, to route auditing events in your account to destinations of your choice by configuring targets and routes that define where activity tracking events are sent. For more information, see About IBM Cloud Activity Tracker Event Routing.

You can use IBM Cloud Logs to visualize and alert on events that are generated in your account and routed by IBM Cloud Activity Tracker Event Routing to an IBM Cloud Logs instance.

As of 28 March 2024, the IBM Cloud Activity Tracker service is deprecated and will no longer be supported as of 30 March 2025. Customers will need to migrate to IBM Cloud Logs before 30 March 2025. During the migration period, customers can use IBM Cloud Activity Tracker along with IBM Cloud Logs. Activity tracking events are the same for both services.

{: important} -->

Locations where activity tracking events are sent to IBM Cloud Logs hosted event search

watsonx Assistant sends activity tracking events to IBM Cloud Logs hosted event search in the regions that are indicated in the following table.

Dallas (us-south)	Washington (us-east)	Toronto (ca-tor)	Sao Paulo (br-sao)
Yes	Yes	No	No

Regions where activity tracking events are sent in Americas locations

Tokyo (jp-tok)	Sydney (au-syd)	Osaka (jp-osa)	Chennai (in-che)
Yes	Yes	No	No

Regions where activity tracking events are sent in Asia Pacific locations

Frankfurt (eu-de)	London (eu-gb)	Madrid (eu-es)
Yes	Yes	No

Regions where activity tracking events are sent in Europe locations

Locations where activity tracking events are sent by IBM Cloud Activity Tracker Event Routing

watsonx Assistant sends activity tracking events by IBM Cloud Activity Tracker Event Routing in the regions that are indicated in the following table.

Dallas (us-south)	Washington (us-east)	Toronto (ca-tor)	Sao Paulo (br-sao)

Yes	Yes	No	No

Regions where activity tracking events are sent in Americas locations

Tokyo (jp-tok)	Sydney (au-syd)	Osaka (jp-osa)	Chennai (in-che)
Yes	Yes	No	No

Regions where activity tracking events are sent in Asia Pacific locations

Frankfurt (eu-de)	London (eu-gb)	Madrid (eu-es)
Yes	Yes	No

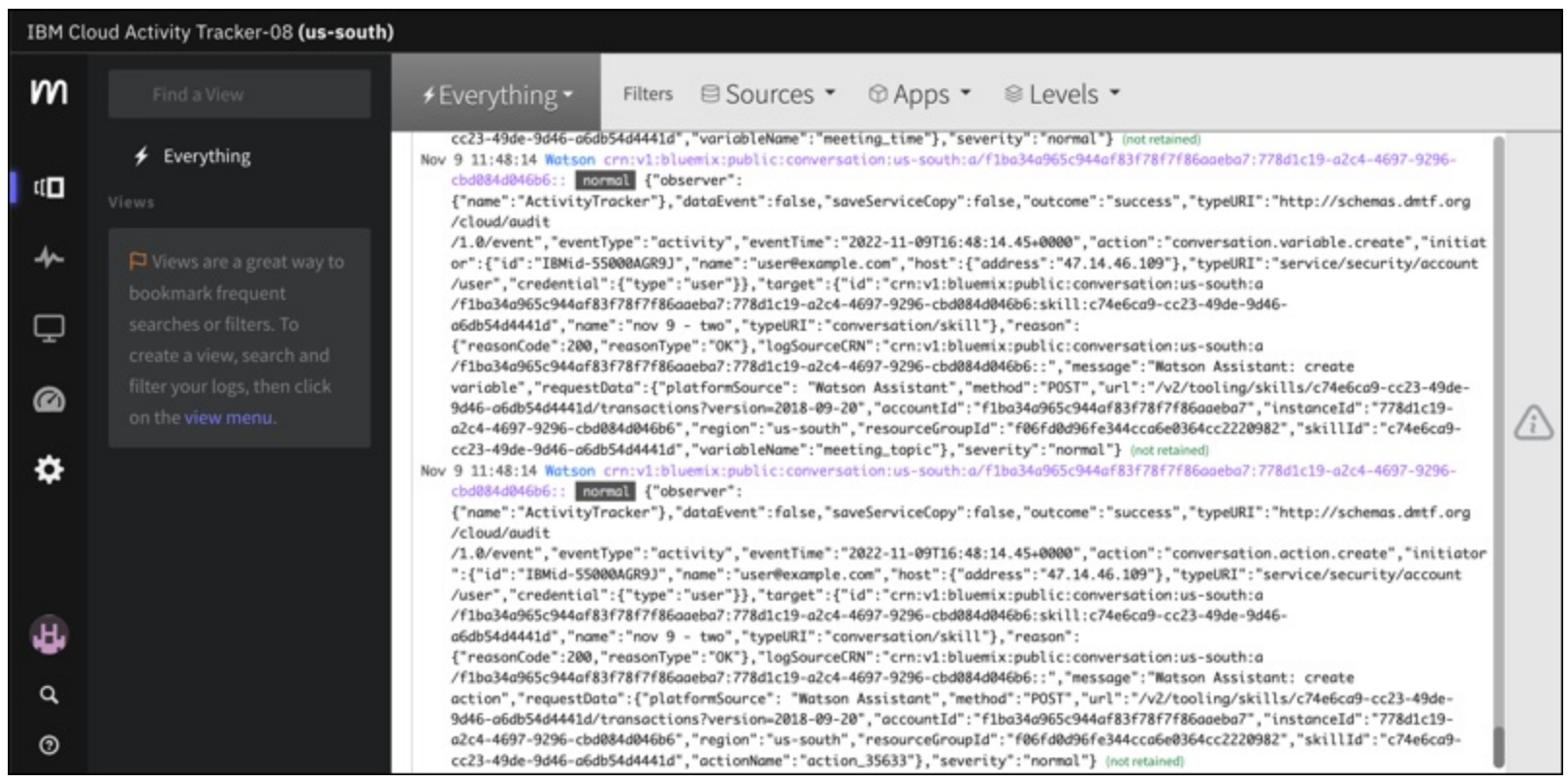
Regions where activity tracking events are sent in Europe locations

Viewing activity tracking events for watsonx Assistant

You can use IBM Cloud Logs to visualize and alert on events that are generated in your account and routed by IBM Cloud Activity Tracker Event Routing to an IBM Cloud Logs instance.

Events that are generated by an instance of the watsonx Assistant service are automatically forwarded to the IBM Cloud Logs service instance that is available in the same location. However, if your service instance is hosted in the **Washington DC** location, create the IBM Cloud Logs service instance in the **Dallas** region.

IBM Cloud Logs can have only one instance per location. To view events, you must access the web UI of the IBM Cloud Logs service in the same location where your service instance is available. For more information, see Navigating to the UI.



Launching IBM Cloud Logs from the Observability page

For information on launching the IBM Cloud Logs UI, see Launching the UI in the IBM Cloud Logs documentation.

List of events

Action	Description
conversation.action.create	creates an action
conversation.action.delete	deletes an action
conversation.action.update	updates an action
conversation.action_handler.create	creates an action handler
conversation.action_handler.delete	deletes an action handler

conversation.action_handler.update	updates an action handler
conversation.action_variable.create	creates an action variable
conversation.action_variable.delete	deletes an action variable
conversation.action_variable.update	updates an action variable
conversation.actions.copy	copies an action from one assistant to another
conversation.assistant.create	creates an assistant
conversation.assistant.delete	deletes an assistant
conversation.assistant.update	updates an assistant, for example, updates the settings
conversation.catalog_integration.create	creates a custom extension
conversation.catalog_integration.delete	deletes a custom extension
conversation.catalog_integration.update	updates a custom extension
conversation.counterexample.create	marks test user input in "Try it out" as being irrelevant or corrects the categorization of a user input that was incorrectly assigned to an intent by marking it as irrelevant
conversation.counterexample.delete	deletes a counterexample
conversation.counterexample.update	edits a counterexample
conversation.data.delete	deletes multiple training data items, such as multiple entities or intents
conversation.data.update	does a bulk action, such as importing a CSV file of intents or entities to the skill
conversation.data_type.create	creates a saved response
conversation.data_type.delete	deletes a saved response
conversation.data_type.update	updates a saved response
conversation.entity.create	creates an entity
conversation.entity.delete	deletes an entity
conversation.entity.update	edits an entity
conversation.environment.create	adds an environment
conversation.environment.delete	deletes an environment
conversation.environment.updates	updates an environment
conversation.example.create	adds a user input example to an intent
conversation.example.delete	deletes a user example from an intent
conversation.example.update	edits a user example that is associated with an intent

conversation.integration_defintion.create	creates an integration
conversation.integration_defintion.delete	deletes an integration
conversation.integration_defintion.update	updates an integration
conversation.intent.create	creates an intent
conversation.intent.delete	deletes an intent
conversation.intent.update	edits an intent
conversation.log.create	corrects an intent that was inaccurately categorized by the skill from the Analytics>User conversations page
conversation.node.create	creates a dialog node
conversation.node.delete	deletes a dialog node
conversation.node.update	edits a dialog node
conversation.notifier.create	creates a notifier
conversation.notifier.delete	deletes a notifier
conversation.notifier.update	updates a notifier
conversation.processor.create	creates a processor
conversation.processor.delete	deletes a processor
conversation.processor.update	updates a processor
conversation.release.create	create a version from content in the draft environment
conversation.release.delete	delete a version
conversation.release.deploy	publish a version to an environment
conversation.skill.create	creates a skill
conversation.skill.delete	deletes a skill
conversation.skill.update	updates a skill
conversation.skill_reference.create	adds a specific skill to an assistant
conversation.skill_reference.delete	removes a specific skill from an assistant
conversation.skill_reference.update	updates a specific skill that is associated with an assistant
conversation.skill_variable.create	create a skill variable
conversation.skill_variable.delete	delete a skill variable
conversation.skill_variable.update	update a skill variable

conversation.skills.export	export a skill
conversation.skills.import	import a skill
conversation.snapshot.create	creates a version of a dialog skill
conversation.snapshot.delete	deletes a version of a dialog skill
conversation.snapshot.update	updates a version of a dialog skill
conversation.step.create	adds a step to an action
conversation.step.delete	deletes a step from an action
conversation.step.update	updates a step in an action
conversation.step_handler.create	create a step handler
conversation.step_handler.delete	delete a step handler
conversation.step_handler.update	update a step handler
conversation.synonym.create	creates a synonym for an entity value
conversation.synonym.delete	deletes a synonym that is associated with an entity value
conversation.synonym.update	edits a synonym that is associated with an entity value
conversation.userdata.delete	deletes data that was created by a specified customer
conversation.value.create	creates an entity value
conversation.value.delete	deletes an entity value
conversation.value.update	edits an entity value
conversation.workspace.create	creates a workspace
conversation.workspace.delete	deletes a workspace
conversation.workspace.update	edits a workspace

Actions that generates events

Additional information for update events

Any of the update events include this additional information in a `requestData.update` object:

- Name changed
- Title changed
- Metadata changed
- Training data changed

This example from an `assistant.update` event shows a name change:

```
$ "update":[{"updateType":"Name changed","nameAttribute":"name","newValue":”Banking Bot 2},{“updateType":"Metadata changed","attributesUpdated":
["description","language"]}],{"environment":"draft"},
```

Securing your assistant

IBM is committed to providing our clients and partners with innovative data privacy, security, and governance solutions.

Notice

Clients are responsible for ensuring their own compliance with various laws and regulations, including the European Union General Data Protection Regulation (GDPR).

Clients are solely responsible for obtaining advice from competent legal counsel to identify and interpret relevant laws and regulations that can affect the clients' business and any actions the clients need to take to comply with such laws and regulations.

The products, services, and other capabilities described herein can have restricted availability and are not suitable for all client situations. IBM does not provide legal, accounting, or auditing advice; or represent or warrant that its services or products can ensure compliance with any law or regulation.

If you need to request GDPR support for IBM Cloud® Watson resources:

- In the European Union, see Requesting support for IBM Cloud Watson resources created in the European Union.
- Outside the European Union, see Requesting support for resources outside the European Union.

You can keep user assistant data secure when interacting with server-run code by configuring Secure Sockets Layer or Transport Layer Security (SSL/TLS) certificates in your assistant. For more information, see Configuring Security certificates.

European Union General Data Protection Regulation (GDPR)

IBM is committed to providing our clients and partners with innovative data privacy, security, and governance solutions to assist them on their journey to GDPR compliance.

Learn more about IBM's own GDPR readiness journey and our GDPR capabilities and offerings to support your compliance journey at IBM Cloud GDPR EU Compliance.

Health Insurance Portability and Accountability Act (HIPAA)

US Health Insurance Portability and Accountability Act (HIPAA) support is available for Enterprise plans that are hosted in the Washington, DC or Dallas locations. For more information, see Enabling HIPAA support for your account.

Do not add personal health information (PHI) to the training data (entities and intents, including user examples) that you create. In particular, be sure to remove any PHI from files that contain real user utterances that you upload to mine for intent or intent user example recommendations.

Configuring Security certificates

You can configure Security certificates in your assistant to ensure secure communication, data protection, and privacy across various applications. By establishing trust and verifying the identity of communicating parties, security certificates help to maintain the confidentiality, integrity, and availability of digital information. You can upload and download certificates, replace old certificates, and delete the certificates in your assistant using the `Security certificates (SSL/TLS)` section.

The assistant supports TLS 1.2 and TLS 1.3, where TLS 1.3 is the default version. If the service does not support TLS 1.3 or 1.2 versions, ask the administrator of that service to provide support for one of these TLS versions.

When your assistant connects to a user-specified URL through custom extensions, conversational skills, or webhooks, it requires verifying the identity of the services to prevent unauthorized access and eliminate security risks. If the user-specified URL uses the Hypertext Transfer Protocol Secure (HTTPS) protocol, your assistant verifies the identity of the secure service in three ways.

`Table 1. Possibilities of identifying secure service` explains the three ways in which the assistant can verify the identity of the secure service when directed to an HTTPS service.

Description	Requirements	Certificate verifier	Security level	Notes
Your assistant has the SSL certificate for the service that is calling.	You provide the SSL certificate for the service to the assistant.	The assistant verifies the identity directly.	Most secure	This is secure even if the SSL certificate is self-signed because signature verification is not needed when the assistant already has the certificate.

Your assistant has the SSL certificate for a certificate authority that has signed the certificate of the service it is calling.	You provide the SSL certificate for the certifying authority to the assistant, and the SSL certificate for the service must be signed by that certifying authority.	The assistant verifies the authenticity of a service through the certificate authority. It knows to trust the certificate authority because you provided the certificate for it.	Extremely secure when the certificate authority is extremely secure	This approach works with private certificate authorities such as the ones that enterprises establish for internal use.
The service has an SSL certificate that is signed by a public trusted certificate authority.	You do not need to provide any SSL certificate to the assistant, but the SSL certificate for the service must be signed by a public trusted authority.	The assistant verifies the authenticity of a service using the public trusted certificate authority.	Extremely secure	Publicly trusted certificate authorities diligently verify the ownership of the private key before signing any certificate.

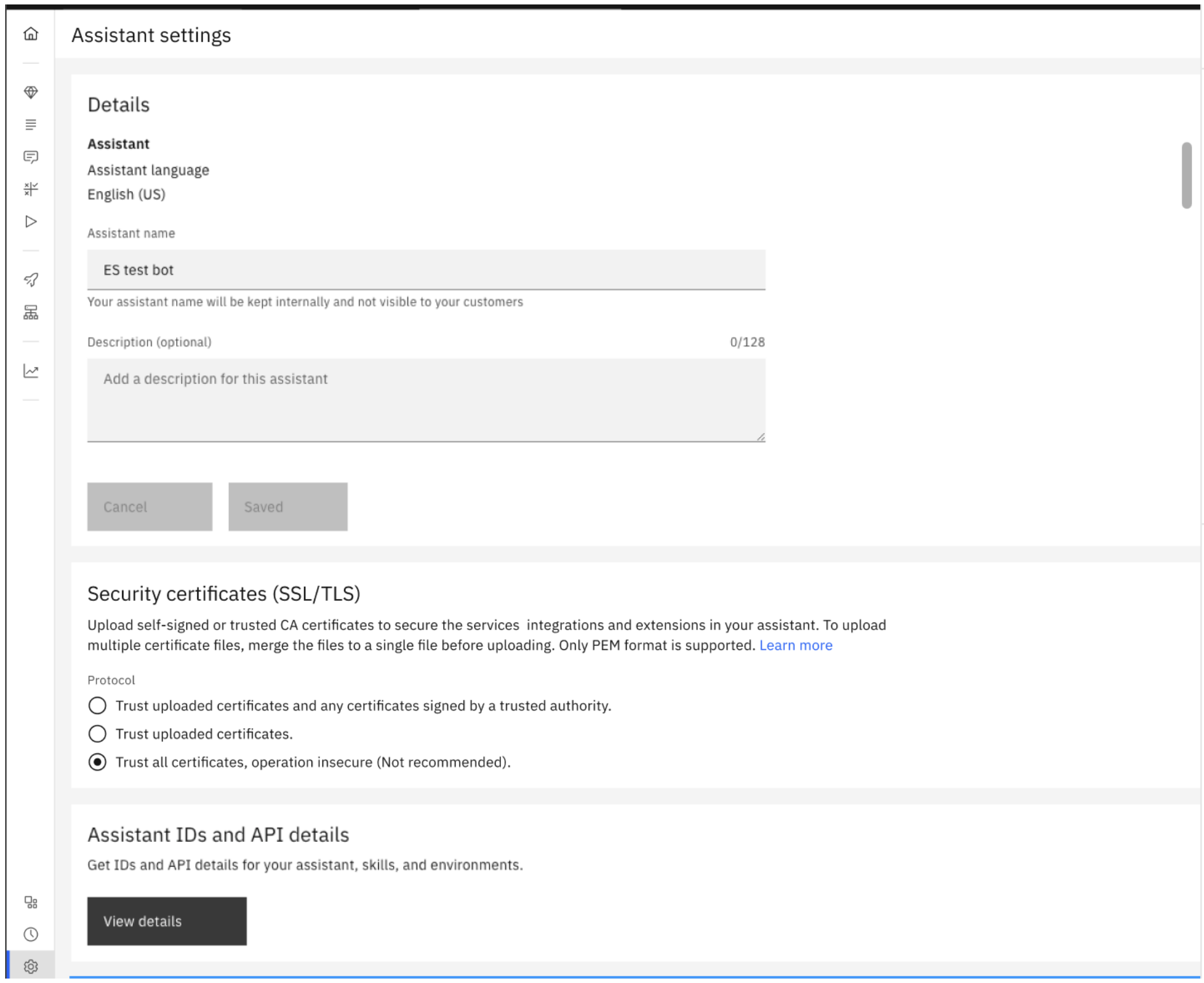
Possibilities of identifying secure service

Some example Security certificates configuration options are discussed with which you can achieve an optimal balance of security and convenience for an assistant with few connection details. For more information, see Methods for choosing the Security certificates option .

Table 2. Security certificates configuration option provides information about the possible Security certificates configuration option in your assistant and its explanation.

Security certificates configuration option	Description	Connectivity	Usage	Uploading option support
Trust uploaded certificates and any certificates signed by a trusted authority	Your assistant can connect to any service: • with a certificate signed by a trusted authority. • with the uploaded certificate (either the certificate for that service or the certificate for its certifying authority). It cannot connect to any other secure service.	• Convenient to use. • You can verify the identity of all services that your assistant connects to. No additional effort is needed for services that have a certificate signed by a publicly trusted authority. • When not signed by a publicly trusted authority, you must upload a certificate file for verifying the identity of the services your assistant connects to.	Commonly used by all assistants.	Yes
Trust uploaded certificates	Your assistant can connect to any service with an uploaded certificate (either the certificate for that service or the certificate for its certifying authority). It cannot connect to any other secure service.	• Less convenient to use. • Your assistant can connect only to secure services with uploaded certificates or their signing authority's certificate.	Less usage	Yes
Trust all certificates, operation insecure (Not recommended)	Your assistant can connect to any service and is not recommended because of lack of protection from imposter services.	Reasonable option • for proof-of-concepts or demo assistants where your assistants don’t have any independent users or sensitive data. • to keep the assistants operational even when they are connected to services that are not signed by publicly trusted authorities.	A default option for both old and new assistants.	No

Security certificates configuration option



SSL certificates configuration option

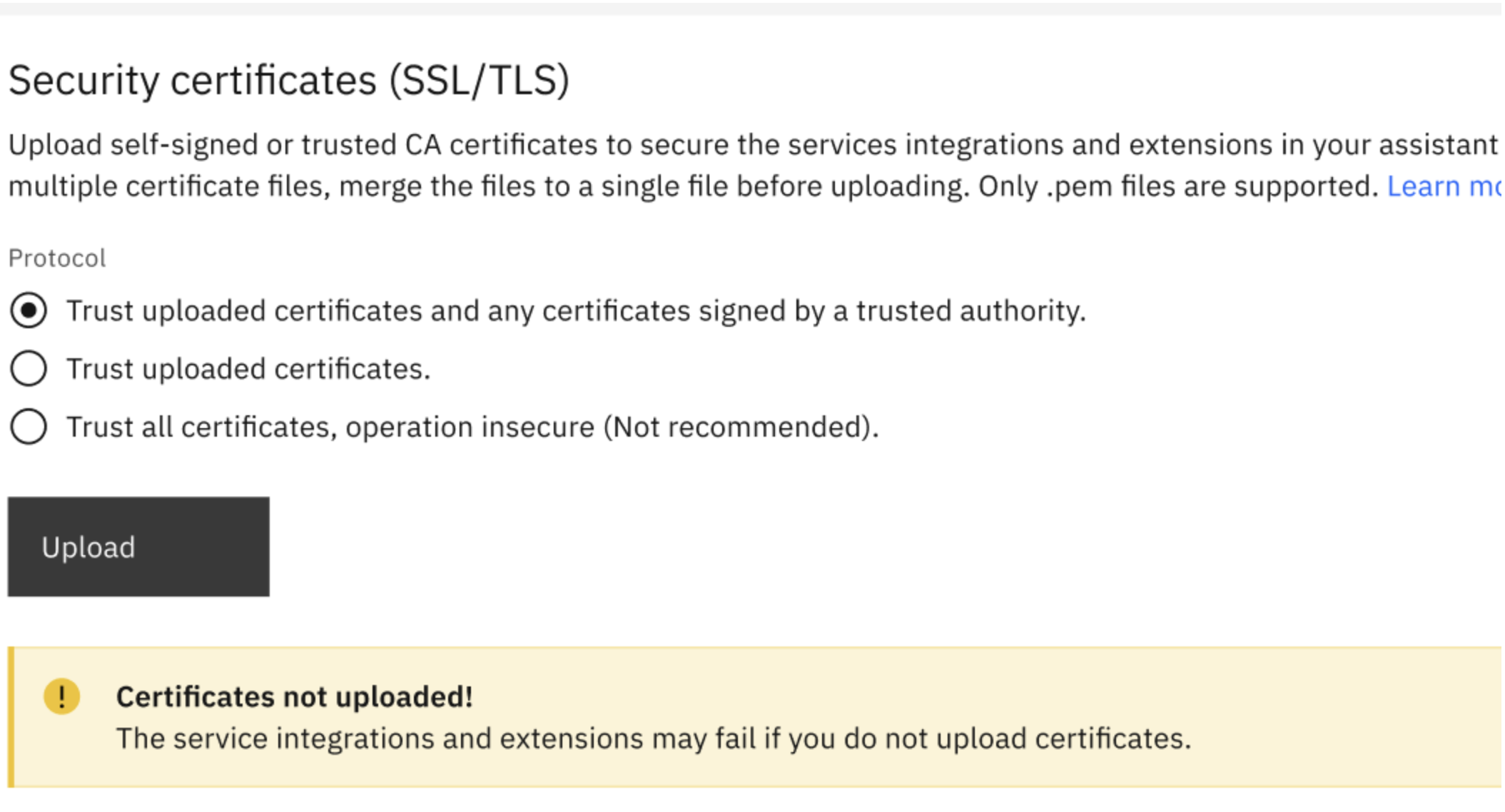
Uploading Security certificates

Before you begin

- Ensure that your files are in the *PEM* format with a maximum size of 500 KB.
- If you have multiple files, merge them to a single file.

To upload self-signed certificates or certificates from a trusted authority in your assistant:

1. Go to **Home** > **Assistant settings**.
2. In the **Security certificates (SSL/TLS)** section, select `Trust uploaded certificates and any certificates signed by a trusted authority` or `Trust uploaded certificates` per your requirement.
3. Click `Upload`.
4. In the `Upload certificate` dialog, add your certificates. You can drag or select the certificate file from your folder.
5. Click `Upload`. You can see the `Secure socket layer connected` message.



Replacing the existing Security certificates

To replace the existing certificates with the new ones in your assistant:

1. Go to **Home** > **Assistant settings**.
2. In the **Security certificates (SSL/TLS)** section, click `Upload`.
3. In the `Upload certificate` dialog, add your new certificates. You can drag or upload the certificate file.

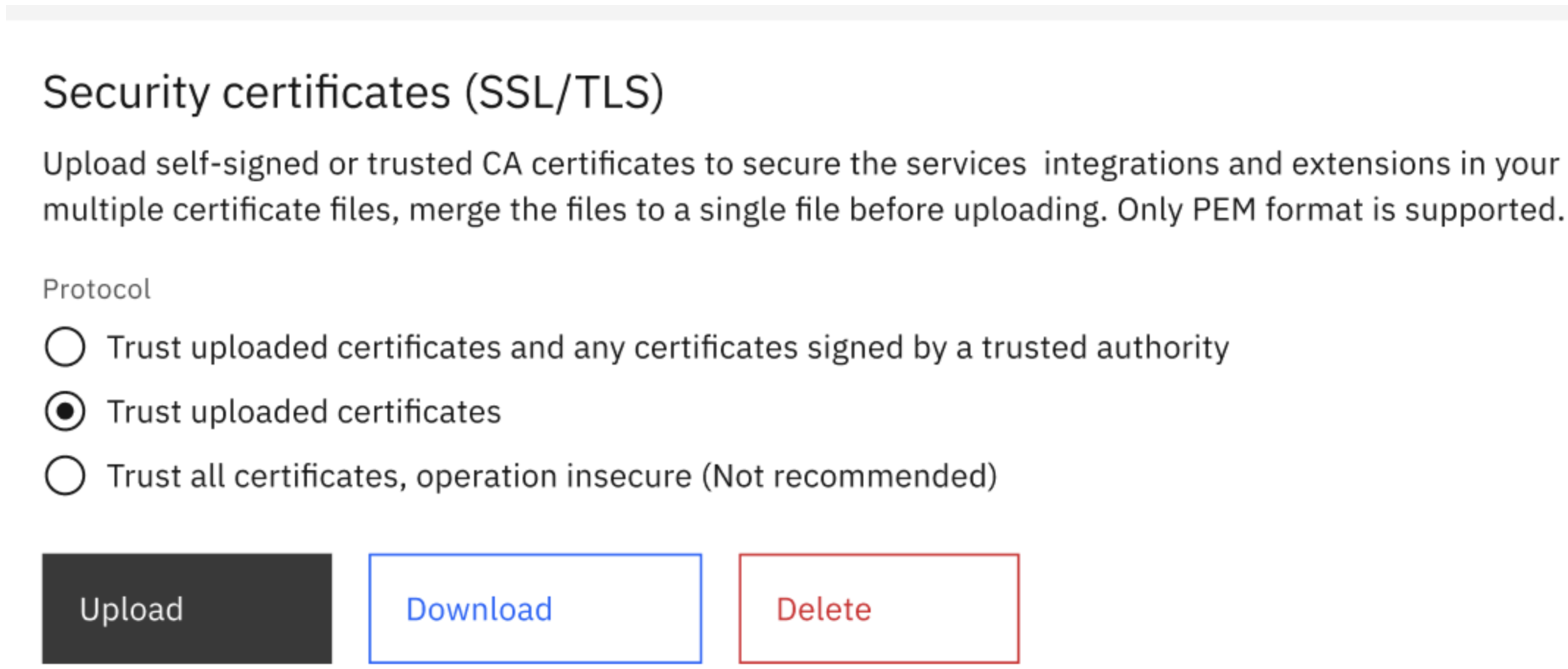
   Note: This option deletes or overwrites the existing certificates.

4. Click `Upload`.

Downloading the Security certificates

To view the existing certificates in your assistant:

1. Go to **Home** > **Assistant settings**.
2. In the **Security certificates (SSL/TLS)** section, click `Download`.
3. You can now open the downloaded file and see the existing certificates.



Modifying the existing Security certificates

To modify the existing certificates in your assistant:

1. Download the Security certificates as described in Downloading the Security certificates.
2. Add or remove certificates from the existing file.
3. Replace the existing *PEM* file with the modified file as described in Replacing the existing Security certificates.

Deleting the existing Security certificates

To delete the existing certificates in your assistant:

1. Go to **Home** > **Assistant settings**.
2. In the **Security certificates (SSL/TLS)** section, click `Delete`.
3. In the **Delete certificate** dialog, select `I acknowledge that I read and understand this warning`.
4. Click `Yes`.

The selected option automatically points to `Trust all certificates, operation insecure (Not recommended)`.

Opting out of log data use

IBM uses log data (Enterprise plan data is excluded) to continually learn from and improve the watsonx Assistant product. The logged data is not shared or made public.

To prevent IBM from using your log data for general service improvements, complete one of the following tasks.

- If you use a custom application, for each API `/message` request, set the `X-Watson-Learning-Opt-Out` header parameter to `true`.

  For more information, see Data collection.

- If you use the web chat integration, add the `learningOptOut` parameter to the script that you embed in your web page, and set it to `true`.

  For more information, see Configuration.

Labeling and deleting data in watsonx Assistant

Do not add personal data to the training data (actions and steps, including user examples) that you create. In particular, be sure to remove any personally identifiable information from files with real user utterances that you upload to mine for user example recommendations.

Experimental and beta features are not intended for use with a production environment and therefore are not guaranteed to function as expected when labeling and deleting data. Experimental and beta features cannot be used to implement a solution that requires the labeling and deletion of data.

If you need to remove a customer's message data from a watsonx Assistant instance, you can use the customer ID of the client if you associate the message with a customer ID when the message is sent to watsonx Assistant.

Note: Removing message data must be an occasional event only for individual customer IDs. To disable analytics logs, you can upgrade to an Enterprise with Data Isolation plan.

- The assistant preview and automatic Facebook integration do not support the labeling or deletion of data based on customer ID, and cannot be used in a solution that must support the ability to delete data based on customer ID.
- For Intercom, the `customer_id` is the `user_id` with an `intercom_` prefix. The Intercom `user_id` property is the `id` of the `author` message object in the Conversation Model that is defined by Intercom.
  - To get the ID, open the channel from a web browser. Open the web developer tools to view the console. Look for `author`.

  The full customer ID looks like this: `customer_id=intercom_5c499e5535ddf5c7fa2d72b3`.
- For Slack, the `customer_id` is the `user_id` with a `slack_` prefix. The Slack `user_id` property is a concatenation of the team ID, such as `T09LVDR7Y`, and the member ID of the user, such has `W4F8K9JNF`. For example, `T09LVDR7YW4F8K9JNF`.
  - To get the team ID, open the channel from a web browser. Open the web developer tools to view the console. Look for `[BOOT] Initial team ID`.
  - You can copy the member ID from the user's Slack profile.
  - To get the IDs programmatically, use the Slack API. For more information, see Overview. The full customer ID looks like this: `customer_id=slack_T09LVDR7YW4F8K9JNF`.
- For the web chat integration, the service takes the `user_id` that is passed in and adds it as the `customer_id` parameter value to the `X-Watson-Metadata` header with each request.

Before you begin

To be able to delete message data associated with a specific user, you must first associate all messages with a unique **customer ID** for each user. To specify the **customer ID** for any messages sent with the `/message` API, include the `X-Watson-Metadata: customer_id` property in your header. For example:

```
$ curl -X POST -u "apikey:3Df... ...Y7Pc9"
 --header
   'Content-Type: application/json'
   'X-Watson-Metadata: customer_id=abc'
 --data
   '{"input":{"text":"hello"}}'
  '{url}/v2/assistants/{assistant_id}/sessions/{session_id}/message?version=2019-02-28'
```

where {url} is the appropriate URL for your instance. For more information, see Service endpoint.

Note: The `customer_id` string cannot include the semicolon (`;`) or equal sign (`=`) characters. You are responsible for ensuring that each `customer ID` property is unique across your customers.

Only the first **customer ID** value that is passed in the `X-Watson-Metadata` header is used as the `customer_id` string for the message log. This **customer ID** value can be deleted with `DELETE /user_data` v1 API calls.

If you add search to an assistant, user input that is submitted to the assistant is passed to the Discovery service as a search query. If the watsonx Assistant integration provides a customer ID, then the resulting `/message` API request includes the customer ID in the header, and the ID is passed through to the Discovery `/query` API request.

To delete query data associated with a specific customer, you must send a separate delete request directly to the Discovery service instance that is linked to your assistant. For more information, see the Discovery information security.

Querying user data

Use the v1 `/logs` method `filter` parameter to search an application log for specific user data. For example, to search for data specific to a `customer_id` that matches `my_best_customer`, the query might be:

```
$ curl -X GET -u "apikey:3Df... ...Y7Pc9" \
"{url}/v2/assistants/{assistant_id}/logs?version=2020-04-01&filter=customer_id::my_best_customer"
```

where {url} is the appropriate URL for your instance. For more information, see Service endpoint.

For more information, see the Filter query reference.

Deleting data

To delete any message log data associated with a specific user that your assistant might store, use the `DELETE /user_data` v1 API method. Specify the customer ID of the user by passing a `customer_id` parameter with the request.

Only data that was added by using the `POST /message` API endpoint with an associated customer ID can be deleted with this delete method. Data that was added by other methods cannot be deleted based on customer ID. For example, entities and intents from customer conversations cannot be deleted this way. Personal Data is not supported for those methods.

IMPORTANT Specifying a `customer_id` deletes *all* messages with this `customer_id`, received before the delete request, across your entire watsonx Assistant instance, not just within one skill.

For example, to delete any user message data that is associated with customer ID `abc` from your watsonx Assistant instance, send the following cURL command:

```
$ curl -X DELETE -u "apikey:3Df... ...Y7Pc9" \
"{url}/v2/user_data?customer_id=abc&version=2020-04-01"
```

where {url} is the appropriate URL for your instance. For more information, see Service endpoint.

An empty JSON object `{}` is returned.

For more information, see the API reference.

Note: Delete requests are processed in batches and can take up to 24 hours to complete.

Web chat usage data

The watsonx Assistant web chat sends limited usage data to the Amplitude service. When a user interacts with the web chat widget, we track features that are being used, and events such as how many times the widget is opened, and how many users start conversations. This information does not include Assistant training data or the content of any chat interactions.

The information sent to Amplitude is not Content as defined in the Cloud Service Agreement (CSA). It is Account Usage Information as described in Section 9.d of the CSA and is handled as described in the IBM Privacy Statement. The purpose of this information gathering is limited to establishing statistics on the use and effectiveness of web chat and making improvements.

Private network endpoints

Plus

You can set up a private network for watsonx Assistant instances that are part of a Plus or Enterprise service plan. A private network prevents data from being transferred over the public internet and ensures greater data isolation.

Private network endpoints support routing services over the IBM Cloud private network instead of the public network. A private network endpoint provides a unique IP address that is accessible to you without a VPN connection.

For more information, see Public and private network endpoints.

Note: Integrations that are provided with the product require endpoints on the public internet. Therefore, any built-in integrations for your assistant have public endpoints.

Related topics

- IBM Cloud compliance programs describe how to manage regulatory compliance and internal governance requirements with IBM Cloud services.

Backing up and restoring data

Back up and restore your watsonx Assistant data by downloading, and then uploading the data.

You can download:

- Actions or dialog from the draft environment
- Published versions of actions or dialog
- Skills from the classic experience

You can upload:

- Actions or dialog to the draft environment

- Skills to the classic experience

Downloading from the draft environment

To download your actions from the draft environment:

1. On the **Actions** page, click **Global settings** ⚙.
2. On the **Upload/Download** tab, click **Download**. The download includes your actions in a JSON file.

To download your dialog from the draft environment:

1. On the **Dialog** page, click **Upload/Download**.
2. On the **Download** tab, click **Download**. The download includes your dialog in a JSON file.

Downloading a published version

You can download a published version from the **Publish** page or from **Assistant settings**.

To download from the Publish page:

1. On the **Publish** page, click the **Download** icon ⤓. for a recent version. The download includes your actions in a JSON file. If you are using dialog, the download also includes your dialog in a separate JSON file.

To download from Assistant settings:

1. Open **Assistant settings**.
2. In the **Download/Upload** section, click **Download/Upload files**.
3. On the **Download** tab, select a published version, then click **Download**. You need at least one version for the download to be available. The .zip file includes your actions in an `action-skill.json` file. If you're using dialog, the .zip file also includes a `dialog-skill.json` file.

   Note: You can also enable a *multilingual download* to download your training and responses in CSV files, which you can use with a translation service. For more information, see Using multilingual downloads for translation.

Downloading a skill from the classic experience

If you are using the classic experience, you can download a skill.

1. On the **Skills** page, locate the skill that you want to download.
2. Click the ⤓ icon, and then choose **Download**.

Uploading to the draft environment

You can upload actions or dialog to the draft environment. The upload replaces any actions or dialog that you're currently working on, so make sure you back up your content or publish a version before you upload.

⚠ **Important:** If the watsonx Assistant service changes between the time you export the actions and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported actions might function differently than before. The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

To upload your actions to the draft environment:

1. On the **Actions** page, click **Global settings** ⚙.
2. On the **Upload/Download** tab, drag a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

To upload your dialog to the draft environment:

1. On the **Dialog** page, click **Upload/Download**.
2. On the **Upload** tab, drag a JSON file onto the tab or click to select a file from your local system, then click **Upload**.

As an alternative, you can use **Assistant settings** to upload to the draft environment:

1. Compress your JSON file into a .zip file. You can include an actions JSON file and a dialog JSON file in the same .zip file if you want.
2. Open **Assistant settings**.
3. In the **Download/Upload** section, click **Download/Upload files**.
4. On the **Upload** tab, choose **Assistant only**.
5. Attach your .zip file package, then click **Upload**.

Uploading a skill to the classic experience

In the classic experience, to reinstate a backup copy of a dialog skill that you exported from another service instance or environment, create a new skill by importing the JSON file of the skill you exported.

> ⚠ **Important:** If the watsonx Assistant service changes between the time you export the skill and import it, due to functional updates that are regularly applied to instances in cloud-hosted continuous delivery environments, your imported skill might function differently than it did before.

1. On the **Skills** page, click **Create skill**.
2. Choose to create a dialog skill, then click **Next**.
3. Click the **Upload skill** tab.
4. Select the JSON file you want to import, then click **Upload**.

   > ⚠ **Important:** The imported JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON file cannot contain tabs, newlines, or carriage returns.

   > **Tip:** The maximum size for a skill JSON file is 10MB. If you need to import a larger skill, consider using the REST API. For more information, see the API Reference.

5. Specify the details for the skill:
   - **Name**: A name no more than 100 characters in length. A name is required.
   - **Description**: An optional description no more than 200 characters in length.
   - **Language**: The language of the user input the skill will be trained to understand. The default value is English.

After you create the skill, it appears as a tile on the Skills page.

Retaining logs

If you want to store logs of conversations, you can use the `/logs` API to export your log data. See API reference for details.

Your service plan determines how long logs are available.

Service plan	Chat message retention
Enterprise with Data Isolation	Last 90 days
Enterprise	Last 30 days
Premium (legacy)	Last 90 days
Plus	Last 30 days
Trial	Last 30 days
Lite	Last 7 days

Log retention by plan

High availability and disaster recovery

IBM® watsonx™ Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.

You are responsible for understanding your configuration, customization, and usage of watsonx Assistant. You are also responsible for being ready to re-create an instance of the service in a new location and to restore your data in any location. See How do I ensure zero downtime? for more information.

High availability

watsonx Assistant supports high availability with no single point of failure. The service achieves high availability automatically and transparently by using the multi-zone region feature provided by IBM Cloud.

IBM Cloud enables multiple zones that do not share a single point of failure within a single location. It also provides automatic load balancing across the zones within a region.

Disaster recovery

Disaster recovery can become an issue if an IBM Cloud location experiences a significant failure that includes the potential loss of data. Because the multi-zone region feature is not available across locations, you must wait for IBM to bring a location back online if it becomes unavailable. If underlying data services are compromised by the failure, you must also wait for IBM to restore those data services.

If a catastrophic failure occurs, IBM might not be able to recover data from database backups. In this case, you need to restore your data to return your service instance to its most recent state. You can restore the data to the same or to a different location.

Your disaster recovery plan includes knowing, preserving, and being prepared to restore all data that is maintained on IBM Cloud. See Backing up and restoring data for information about how to back up your service instances.

Failover options

Learn about options that you can use to increase the availability of watsonx Assistant for your organization.

Introduction

IBM® watsonx™ Assistant instances are deployed across multizone regions (MZRs) in all data centers (except Seoul, Korea) and can withstand the loss of any individual zone within an MZR. However, regional outages can occur and while IBM strives to keep outages to a minimum, you might want to consider moving traffic to a different region for your business-critical applications.

You should have a copy of your watsonx Assistant instance in a different MZR (for example, deploy your assistant in US East and US South). You must purchase a second service instance of watsonx Assistant, and second instances of the necessary assistants and skills need to be instantiated. You must implement change management controls to synchronize the changes between the two regions. watsonx Assistant and Speech services have APIs available to export and import the definitions.

With two instances, consider two topologies:

- **Active/active**: Both instances always serving traffic with the load sprayed between the two.
- **Active/passive**: One instance is active and the passive site receives traffic only if a failover happens.

Each approach has pros and cons. Considerations specific to watsonx Assistant are detailed in the sections that follow.

Considerations

IBM® watsonx™ Assistant instances in one region are unaware of instances in a second region, which can affect some features and capabilities in watsonx Assistant.

Analytics

IBM® watsonx™ Assistant analytics provide overview statistics on the number of interactions with users and containment rates. Analytics doesn't cumulate statistics across regions. With an active/passive topology, this approach to analytics is sufficient. However, using an active/active topology likely requires you to use webhooks to gather interaction data, and build custom data warehouses and reports to understand total usage.

Session history for web chat and the v2 api

Session history allows your web chats to maintain conversation history and context when users refresh a page or change to a different page on the same website. This feature doesn't work across instances, so in-progress conversations need to be restarted.

Billing

IBM calculates your bill based on the IBM Cloud Account. IBM® watsonx™ Assistant calculates monthly average user (MAU) metrics by aggregating within a specific service instance as follows:

- The same MAU used in 2 different assistant resources in the **same** service instance counts as 1 MAU
- The same MAU used in 2 different assistant resources in **different** service instances counts as 2 MAUs

For an **active/active** topology, under the worst case scenario, the MAU count might end up being doubled for a billing period.

Phone integration

A phone integration in one region is unaware of a phone integration in a different region. You need to ensure that your assistants are identically configured in both regions. You also need to rely on the upstream SIP trunking provider to detect and manage failing over between regions.

Monitoring

SIP trunking providers can be configured to actively health-check the watsonx Assistant session border controllers (SBCs) by sending periodic SIP OPTIONS messages to each zone within a region. A failure to receive a response can be used to either provide notification of a failure to trigger a manual failover, or to automate removal of the failed zone from the route list.

Failover

The SIP trunking provider plays an important role in detecting and managing a failover, especially if an automatic failover is expected between regions. Usually, SIP trunking providers should be configured to treat each zone within a region as active/active and two regions where an assistant is configured as active/passive. SIP trunking providers should always be configured to balance the load and fail over between zones within a single region.

Full outage

Phone integration failures have two types. The first type is a full outage where the session border controllers in all 3 regional zones become unreachable. This type of outage is easier to detect and handle because the SIP trunking provider is immediately notified by SIP timeouts that the call fails and can be configured to either automatically fail over or the call routing can be manually reconfigured at the SIP trunking provider to direct traffic away from the failed region toward the passive backup region. If a failover is automated and a regional backup is enabled, it is always best to try a different zone first and redirect traffic to the passive backup region only if a preconfigured number of failures occur within a short period. This prevents an unnecessary failover between regions if only a short outage occurs.

IBM® watsonx™ Assistant provides a round-robin fully qualified domain name (FQDN) that includes the IP addresses for each zone in the region. Many SIP trunking providers automatically retry each IP in the FQDN when failures occur. To support disaster recovery, the service provider might need to configure two separate SIP trunks, one for each region, and only when all the zones in a single region fail should the call be switched to the backup region. It's important to set the SIP INVITE failure timeouts at the SIP trunking provider low enough to avoid long call setup latencies when a failover is occurring.

Partial outage

The second type of failure is a partial service outage within the region. A partial outage is much harder to detect and manage because of the large number of variations in service failures that can occur within a region. Sometimes, small issues affect the performance characteristics of the call but not cause the call to fail.

For issues that ultimately cause a call to fail, there are two ways your assistant can handle the call. The first is to accept the call and then transfer it to a configured default SIP URI. You can configure this setting in the phone integration and is also used for mid-call failures. The default transfer target SIP URI is defined in the **SIP target when a call fails** field that is on the Advanced tab of the phone integration configuration.

The phone integration can also be configured to respond to a SIP INVITE with a SIP 500 (service unavailable) message if an outage is detected during call setup instead of transferring a call to a live agent. A SIP 500 can then be used to redirect the call to another zone, or if many SIP 500s are received, to another region. Using a SIP 500 INVITE error is a better way to signal a failure to an upstream SIP trunking provider because it gives the provider a way to reroute the call. Using only the default transfer target to handle call failures is acceptable for low call volume scenarios but can result in large numbers of calls that are directed to a customer contact center when bigger call volumes are handled by your assistant. To enable this 500 error response capability for a specific instance, you need to make a request to IBM.

You should plan for both full and partial service outages. A good first step is to plan for a manual failover between regions before you enable automation. You need a complete replica of watsonx Assistant in both regions, including all custom speech model training. When automation is enabled, it is best to start with a strategy for detecting and failing over to the passive backup region when a complete regional outage is detected. After implementation, develop a strategy to deal with partial outages, which should cover most of the failure conditions that can occur with a phone integration deployment.

Web chat

Monitoring

Web chat provides an `onError` listening feature that allows the host page to detect specific types of outage errors, in particular INITIAL_CONFIG,

OPEN_CONFIG, and MESSAGE_COMMUNICATION errors.

For more information, see Listening for errors.

Failover

Handling a failover for web chat is simple, assuming you set up an extra web chat integration in another region. When the failover needs to be manually triggered, make the following changes:

- The embed script that contains your integration ID, region, service instance ID, and subscription ID (if applicable) needs to be changed or updated to use the IDs for the new integration and region.
- If you are using Salesforce or ZenDesk integrations for connecting to human agents, update the configuration within those systems to make sure they can communicate with the correct integration. Follow the instructions on the **Live agent** tab in the web chat configuration for setting up those systems. This capability is only needed for obtaining the conversation history for the agent.
- If you enable web chat security and you are using encrypted payloads, the IBM-provided public key that is used for the encryption might be different depending on the region. If so, you need to update the system that generates the JSON Web Token (JWT) to use the correct key.

You can set up an active/active configuration of web chat by using different integration IDs in your embed scripts, and by ensuring the web chat integration is "sticky" by user. Otherwise, if the user fails over to a different integration, the conversation history might be lost.

API

Monitoring

Capture and monitor response codes of `/message` calls. Response codes of live `/message` traffic should be captured and aggregated.

Ideally, save the response code statistics in an external persistence store. This shared store can aggregate response code results from all deployed instances of your application and provide the greatest fidelity in determining whether a failover should be triggered.

Alternatively, response codes can be aggregated and evaluated in memory for an instance of your application. As only a fraction of traffic is contributing to the failover decision, this might affect how often the failover decision is acted upon.

With either aggregation approach, exceeding a defined threshold might trigger a failover. Common approaches for determining a failover include:

- Percentage-based: greater than X% of requests return a non-200 response code
- Consecutive-based: X calls in a row return a non-200 response code
- Limit-based: X calls return a non-200 response in a time frame

To avoid an unnecessary failover between regions, make sure that a robust retry mechanism is present when calling the `/message` endpoint.

Failover for v1 and v2 stateless API

For a successful failover:

- Training data changes should be synchronized across regions. Avoid pushing changes delayed over a large window of time (such as days) to mitigate the risk of algorithm changes that are deployed by watsonx Assistant in between regions being updated.
- The same IBM Cloud account should be used for the service instances across regions to maintain a single overall bill for services.
- The client applications should support:
  - IBM® watsonx™ Assistant API hostname
  - Service instance credentials
  - v1: workspace_id
  - v2: assistant_id

Although it does not affect the runtime flow of calling `/message`, if you are using fine-grained access control that uses IBM Identity and Access Management (IAM), make sure that the IAM policies are synchronized across the regions. IAM is a global service, but the custom resources (assistants and skills) used by watsonx Assistant access control means each region, which has specific resources, requires specific policies.

For an **active/passive** topology, some form of a circuit break pattern can be used. A single service instance in a region is used exclusively unless errors are detected. At that point, the system can respond by updating the relevant failover metadata to route traffic to the service instance in the other region. When a failover happens, you can decide to continue using the new region as the active instance, or if you want to resume using the initial region when it stabilizes.

For an **active/active** topology, some form of a load balancing can be used, where two or more service instances in different regions always receive a percentage of traffic. Extra logic would need to be established to determine when to pull a region out of rotation. This monitoring logic might use a circuit break pattern similar to the active/passive configuration or rely on a separate dedicated monitoring framework that determines region health. Similar to

active/passive, determining when to insert a region back in rotation would need to be considered as well.

Failover for v2 stateful API

Failover for the v2 stateful API is similar to stateless, with these details to consider:

- The state of a conversation is persisted by watsonx Assistant in a database that is tied to a particular region. As such, a failover for the stateful v2 `/message` may be more disruptive.
- For an **active/passive** topology, you should assume that all in-progress conversations are ended.
- For an **active/active** topology, given the region-locked persistence constraints of the v2 stateful `/message` architecture, all turns (`/message` API calls) of a conversation (session) should occur within the same region.

Adding support for global audiences

Your customers come from all around the globe. You need an assistant that can talk to them in their own language and in a familiar style. Choose the approach that best fits your business needs.

- Quickest solution: The simplest way to add language support is to author the assistant in a single language. You can translate each message that is sent to your assistant from the customer's local language to the assistant language. Later you can translate each response from the assistant language back to the customer's local language.

  This approach simplifies the process of authoring and maintaining the conversation. You can build one assistant and use it for all languages. However, the intention and meaning of the customer message can be lost in the translation.

  For more information about webhooks you can use for translation, see Webhook overview.

- Most precise solution: If you have the time and resources, the best user experience can be achieved when you build multiple assistants, one for each language that you want to support. IBM® watsonx™ Assistant has built-in support for all languages. Use one of 13 language-specific models or the universal model, which adapts to any other language you want to support.

  When you build an assistant that is dedicated to a language, a language-specific classifier model is used by the assistant. The precision of the model means that your assistant can better understand and recognize the goals of even the most colloquial message from a customer.

  Use the universal language model to create an assistant that is fluent even in languages that watsonx Assistant doesn't support with built-in models.

  To deploy, use the web chat integration with your French-speaking assistant to deploy to a French-language page on your website. Deploy your German-speaking assistant to the German page of your website. Maybe you have a support phone number for French customers. You can configure your French-speaking assistant to answer those calls, and configure another phone number that German customers can use.

  You can enable the download of language data files, in CSV format, so you can translate training examples and assistant responses from English into other languages and use in other assistants. For more information, see Using multilingual downloads for translation.

Understanding the universal language model

An assistant that uses the universal language model applies a set of shared linguistic characteristics and rules from multiple languages as a starting point. It then learns from the training data that you add to it.

The universal language classifier can adapt to a single language per assistant. It cannot be used to support multiple languages within a single assistant. However, you can use the universal language model in one assistant to support one language, such as Russian, and in another assistant to support another language, such as Hindi. The key is to add enough training examples or intent user examples in your target language to teach the model about the unique syntactic and grammatical rules of the language.

Use the universal language model when you want to create a conversation in a language where no model is available, and which is unique enough that an existing model is insufficient.

As you follow the normal steps to design a conversational flow, you teach the universal language model about the language you want your skill to support. It is by adding training data that is written in the target language that the universal model is constructed.

For more information about feature support in the universal language model, see Supported languages.

Integration considerations

Keep these tips in mind for integrations:

- Web chat: Web chat has some hardcoded strings that you can customize to reflect your target language. For more information, see Supporting global audiences in web chat.
- Phone integration: If you want to deploy an assistant that uses the universal language model, you must connect to custom Speech service language

models that can understand the language you're using. For more information about supported language models, see the Speech to Text and Text to Speech documentation.

- Search integration: If you build an assistant that specializes in a single language, be sure to connect it to data collections that are written in that language. For more information about the languages that are supported by Discovery, see Language support.

Supported languages

IBM® watsonx™ Assistant supports individual features to varying degrees per language. It has classifier models that are designed specifically to support conversations in the following languages:

Language	Code
English	`en-us`
Arabic	`ar`
Chinese (Simplified)	`zh-cn`
Chinese (Traditional)	`zh-tw`
Czech	`cs`
Dutch	`nl`
French	`fr`
German	`de`
Italian	`it`
Japanese	`ja`
Korean	`ko`
Portuguese (Brazilian)	`pt-br`
Spanish	`es`
Universal*	`xx`

*If you want to support conversations in a language for which watsonx Assistant does not have a dedicated model, such as Russian, use Universal.

Changing an assistant language

After an assistant is created, its language cannot be modified.

Working with accented characters

In a conversational setting, users might or might not use accents with the watsonx Assistant service. As such, both accented and non-accented versions of words might be treated the same for intent detection and entity recognition.

However, for some languages like Spanish, some accents can alter the meaning of the entity. Thus, for entity detection, although the original entity might implicitly have an accent, your assistant can also match the non-accented version of the same entity, but with a slightly lower confidence score.

For example, for the word "barrió", which has an accent and corresponds to the past tense of the verb "barrer" (to sweep), your assistant can also match the word "barrio" (neighborhood), but with a slightly lower confidence.

The system provides the highest confidence scores in entities with exact matches. For example, `barrio` isn't detected if `barrió` is in the training set; and `barrió` isn't detected if `barrio` is in the training set.

You are expected to train the system with the proper characters and accents. For example, if you are expecting `barrió` as a response, put `barrió` into the training set.

Although not an accent mark, the same applies to words that use the Spanish letter `ñ` versus the letter `n`, such as "uña" versus "una". In this case, the letter `ñ` is not an `n` with an accent; it is a unique, Spanish-specific letter.

Using multilingual downloads for translation

You can enable the download of language data files, in CSV format, so you can translate training examples and assistant responses into other languages and use in other assistants.

Each CSV file includes `translatable_string` data that you can use with a machine or human translation service.

Each CSV file also includes `id`, `resource_type`, and `locator` data that watsonx Assistant can use in another assistant to re-create your source assistant. You don't need to edit this information.

To successfully download the multilingual package, the total number of translation entries for both action and dialog skills in your assistant must be less than or equal to 400000. You can configure the limit to a higher value in the Enterprise with Data Isolation plan and IBM® watsonx™ Assistant on-premises. Higher values require an additional resource allocation to support large assistants.

To configure the translation entry limit to a higher value, contact IBM Support Center

The overview of the multilingual process is:

- Enable multilingual download: In your source assistant, enable multilingual download
- Translate content: Use the CSV files with a translation service
- Upload to language-specific assistants: In a destination assistant for another language, use the CSV files to upload translated training and responses

Enabling multilingual download

To enable multilingual download:

1. Open Assistant settings.
2. In the Download/Upload section, click Enable multilingual download.

   > **Note:** Enabling multilingual might take a few minutes to process, but you can work elsewhere in the assistant. The Download/Upload button is disabled until this process finishes. After the multilingual download is enabled in an assistant, it can't be disabled.

3. Click Download/Upload files.
4. On the Download tab, choose Multilingual file package.
5. Select a published version, then click Download. You need at least one version for the download to be available.

   > **Note:** Downloading your first file might take a few minutes to process, but you can work elsewhere in the assistant. The Download/Upload button is disabled until this process finishes.

6. When the download finishes, your .zip file contains:
   - `action-responses.csv`
   - `action-training.csv`
   - `Data (Do not edit)` folder, which contains `assistant.bin`

   If you are using dialog, the .zip file also contains:
   - `dialog-responses.csv`
   - `dialog-training.csv`

Translating content

To translate content:

1. Translate the content in the CSV files, for example, from English to French. Save files with translations as CSV UTF-8 (Comma Delimited) to account for any language-specific characters.
2. When you are finished with the translation process, create a new .zip file that includes:

- `action-responses.csv` with your translations. Don't change the name of the file.
- `action-training.csv` with your translations. Don't change the name of the file.
- The original, untouched `Data (Do not edit)` folder, which contains `assistant.bin`

If you translated dialog content, also include `dialog-responses.csv` and `dialog-training.csv` in the .zip file.

Uploading to language-specific assistants

To upload to a language-specific assistant:

1. Create or switch to a destination assistant that uses the language for your translations.
2. In the destination assistant, open Assistant settings.
3. If you translated dialog training and responses, ensure that dialog is active in the destination assistant.
4. In the Download/Upload section, click Download/Upload files.

   Note: You don't need to enable multilingual download in the destination assistant.

5. On the Upload tab, choose Multilingual file package.
6. Attach your multilingual .zip file package, then click Upload. The translated content is added to your draft environment, so you can work on publishing the translated assistant.

Content language support

These languages are supported for content in actions, dialog, and the search integration.

Language	Actions	Dialog	Search integration
English (`en`)	✓	✓	✓
Arabic (`ar`)	✓	✓	✓
Chinese (Simplified) (`zh-cn`)	✓	✓	✓
Chinese (Traditional) (`zh-tw`)	✓	✓	✓
Czech (`cs`)	✓	✓	✓
Dutch (`nl`)	✓	✓	✓
French (`fr`)	✓	✓	✓
German (`de`)	✓	✓	✓
Italian (`it`)	✓	✓	✓
Japanese (`ja`)	✓	✓	✓
Korean (`ko`)	✓	✓	✓
Portuguese (Brazilian) (`pt-br`)	✓	✓	✓
Spanish (`es`)	✓	✓	✓
Universal (`xx`)	✓	✓	✓

Content support details

Note: The watsonx Assistant service supports multiple languages as noted, but the user interface itself (such as descriptions and labels) is in English. All supported languages can be input and trained through the English interface.

GB18030 compliance: GB18030 is a Chinese standard that specifies an extended code page for use in the Chinese market. This code page standard is important for the software industry because the China National Information Technology Standardization Technical Committee mandates that any software application that is released for the Chinese market after September 1, 2001, be enabled for GB18030. The watsonx Assistant service supports this encoding, and is certified GB18030-compliant

Dialog language support

For these dialog features, language support differs depending on the language.

- Intent feature support
- User input processing support
- Entity feature support

Intent feature support

Language	Content Catalog	Algorithm version
English (en)	✓	✓
Arabic (ar)	✓ (except Covid-19)	✓
Chinese (Simplified) (zh-cn)		✓
Chinese (Traditional) (zh-tw)		✓
Czech (cs)		✓
Dutch (nl)		✓
French (fr)	✓	✓
German (de)	✓ (except Covid-19)	✓
Italian (it)	✓ (except Covid-19)	✓
Japanese (ja)	✓ (except Covid-19)	✓
Korean (ko)		✓
Portuguese (Brazilian) (pt-br)	✓	✓
Spanish (es)	✓	✓
Universal (xx)		

Intent feature support details

User input processing support

Language	Dictionary-based entity support	Fuzzy matching (*Misspelling*)	Fuzzy matching (*Stemming*)	Fuzzy matching (*Partial match*)	Autocorrection
English (en)	✓	✓	✓	✓	✓
Arabic (ar)	✓	✓			

Chinese (Simplified) (zh-cn)	✓				
Chinese (Traditional) (zh-tw)	✓				
Czech (cs)	✓	✓	✓		
Dutch (nl)	✓	✓			
French (fr)	✓	✓	✓		Beta
German (de)	✓	✓	✓		
Italian (it)	✓	✓			
Japanese (ja)	✓	✓			
Korean (ko)	✓	✓			
Portuguese (Brazilian) (pt-br)	✓	✓			
Spanish (es)	✓	✓			
Universal (xx)	✓	✓			

User input processing support details

Entity feature support

Language	Contextual entities	System entities
English (en)	✓	✓
Arabic (ar)		✓
Chinese (Simplified) (zh-cn)		✓
Chinese (Traditional) (zh-tw)		✓
Czech (cs)		✓
Dutch (nl)		✓
French (fr)	Beta	✓
German (de)		✓
Italian (it)		✓
Japanese (ja)		✓
Korean (ko)		✓
Portuguese (Brazilian) (pt-br)		✓

Spanish (es)	✓
Universal (xx)	✓

Entity feature support details

Switching between watsonx Assistant and the classic experience

You can easily switch back and forth between watsonx Assistant and the classic experience. However, watsonx Assistant provides a simplified user interface, an improved deployment process, and access to the latest features. For more information, see Migrating to watsonx Assistant.

To switch between the new and classic experiences, following these steps:

1. From the watsonx Assistant interface, click the **Manage** icon to open your account menu.
2. Select **Switch to classic experience** or **Switch to new experience** from the account menu.

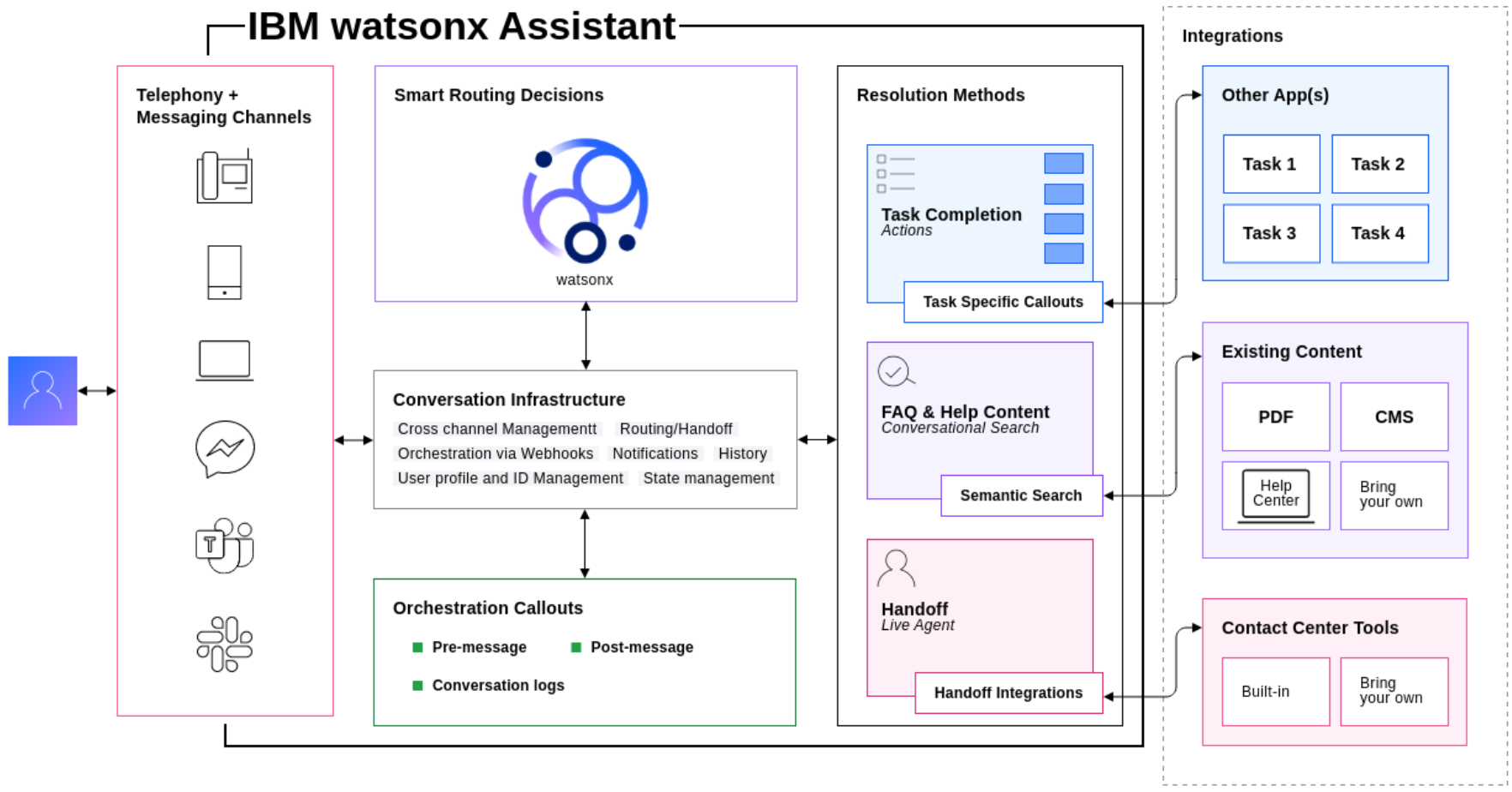
You won't lose any work if you switch to a different experience, and other users of the same instance are not affected. However, keep in mind that any work you do in one experience is not available in the other experience. You can switch back and forth at any time.

Customizing and developing

Overview: Customizing and developing

The watsonx Assistant user interface makes it easy to build an assistant and deploy it to your customers without writing any code. For advanced users and developers, you can further customize and extend the capabilities of your assistant.

A deployed assistant includes numerous components that work together to deliver the help your customers need over the channels they use.



Architecture

- Customers interact with the assistant by using a **channel** such as the web chat or phone integration.
- Based on natural language understanding, the assistant decides to route the customer's request to the appropriate resolution mechanism, which might be an action or a search of existing content.
- The assistant might also need to communicate with external services or hand off the conversation to a human agent.

There are multiple points at which a developer can customize and extend how the assistant behaves, or how it interacts with external services. These customization points include:

- Customizing actions: By writing expressions and editing JSON data, you can extensively customize how an action evaluates step conditions, how it stores data, how it responds to customer input, and how it interacts with channels.
- Web chat development overview: You can use the web chat API to extensively customize the appearance and behavior of the web chat.
- Customizing the phone integration: You can use commands and context variables to extensively configure how your assistant interacts with customers that use the phone integration.
- Customizing the SMS integration: You can use commands and context variables to customize how your assistant interacts with customers that use text messages.
- Extending your assistant using webhooks: You use webhooks to call external services that extend the capabilities of your assistant or log activity.
- Developing a custom channel: If none of the built-in channel integrations meet your needs, you can use the watsonx Assistant REST API and SDKs to develop a custom client application that interacts with your assistant.

Configuring Base LLM

watsonx Assistant includes a range of pre-built AI models and algorithms that can be used to generate text, images, and other types of content. These models can be customized and combined in various ways to create more sophisticated and powerful generative AI applications.

The **Base large language model (LLM)** section in the **Generative AI** page helps you to configure large language models for your assistants. The LLMs enable your customers to interact with the assistants seamlessly without any custom-built conversational steps. You can enable the Base LLM features for the existing actions in your assistants that improve their conversation capability.

You can do the following actions in the Base LLM configuration:

Selecting a large language model for your assistant

To select the LLM that suits your enterprise ecosystem, do the following steps:

1. Go to **Home** > **Generative AI**.
2. In the **Base large language model (LLM)** section, select the large language model from the **Select a model** dropdown.

LLM description table

The following table shows the list of LLMs supported by the watsonx Assistant.

LLM model	Description
ibm/granite-3-8b-instruct	Granite-3.0-8B-Instruct is a 8B parameter model finetuned from Granite-3.0-8B-Base using a combination of open source instruction datasets with permissive license and internally collected synthetic datasets. The model is designed to respond to general instructions and can be used to build assistants for multiple domains, including bussiness applications.
ibm/granite-3-2b-instruct	Granite-3.0-2B-Instruct is a lightweight and open-source 2B parameter model fine tuned from Granite-3.0-2B-Base on a combination of open-source and proprietary instruction data with permissive license. The model is designed to respond to general instructions and can be used to build assistants for multiple domains, including business applications.

LLMs in Assistant

Adding prompt instructions

You can instruct the LLM in your assistant to give refined responses by adding prompt instructions. The prompt instructions help LLMs to guide the conversations with clarity and specificity to achieve the end goal of an action. Follow these steps to add the prompt instruction:

1. Go to **Home** > **Generative AI**.
2. In the **Add prompt instructions** section, click **Add instructions** to see the **Prompt instruction** field.
3. Enter the prompt instructions in the **Prompt instruction** field.

   The maximum number of characters that you can enter in the **Prompt instruction** field is 1,000.

Selecting the answering behavior of your assistant

You can configure the answering behavior of your assistant to provide responses that are based on the preloaded content. The answering behavior that you can configure is:

- **Conversational search**

To use the conversational search, configure search integration and enable conversational search.

Note: Toggling off: Conversational search disables the process that calls it in the routing priority path. You are not disabling the search capability itself.

Important: Only Plus or Enterprise plans support Conversational search. Starting from June 1, 2024, add-on charges are applicable for using the Conversational search feature in addition to your Plus or Enterprise plans. For more information about the pricing plans, see Pricing plans. For more information about terms, see Terms.

In conversational search, the LLM uses content that is preloaded during the search integration to respond to customer queries.

Languages supported for conversational search

You can use conversational search with the languages other than English, including French, Spanish, German, and Brazilian Portuguese. You can test to

verify that you are getting reasonable results in your language.

Deprecated and withdrawn models

The IBM® watsonx™ Assistant product is evolving continuously and as technology improves, some foundation models get deprecated or withdrawn from support. Each model contains the reference for its **Model card** where you can get more information about technical specifications of the model.

Deprecated foundation models

Deprecated

The "Deprecated" tag indicates that the foundation model is deprecated. You can continue using the deprecated foundation model, but you might receive a notification in the UI about the upcoming model removal.

See the following table for the complete information about the deprecated models and a guidance to the next recommended model:

Foundaton model name	Model card	Deprecated	Next recommended model
granite-13b-chat-v2	ibm/granite-13b-chat-v2	November 2024	granite-3-8b-instruct

Foundation models that are deprecated.

Withdrawn foundation models

Withdrawn

The "Withdrawn" tag indicates that the support to a foundation model is withdrawn, and the product moves you to the most recent version of the same model family.

See the following table for the complete information about the withdrawn models and a guidance to the next recommended model:

Foundation model name	Model card	Withdrawn	Next recommended model
granite-13b-chat-v2	ibm/granite-13b-chat-v2	From 19 January 2025	granite-3-8b-instruct

Foundation models that are withdrawn.

Defining responses by using the JSON editor

In some situations, you might need to define your assistant's responses by using the JSON editor. For more information, see Adding assistant responses.

To edit a response by using the JSON editor, click the **Switch to JSON editor** icon </> in the **Assistant says** field. The JSON editor shows how the response is defined behind the scenes and sent to the channel.

Generic JSON format

If you open the JSON editor on a new, empty response, you see the following basic structure:

```
{
  "generic": []
}
```

The `generic` property defines an array of responses that are sent to the channel when the step is executed. The term *generic* refers to the fact that these responses are defined by using a generic JSON format that is not specific to any channel. This format can accommodate various response types that are supported by multiple integrations, and can also be implemented by a custom client application that uses the REST API.

The `generic` array for a step can contain multiple responses, and each response has a *response type*. A basic step that sends a simple text response typically includes only a single response with the response type `text`. However, many other response types are available, supporting multimedia and interactive content, plus control over the behavior of some channel integrations.

Although the `generic` format can be sent to any channel integration, not all channels support all response types, so a particular response might be ignored or handled differently by some channels. For more information, see see Response types reference.

Note: At run time, output with multiple responses might be split into multiple message payloads. The channel integration sends these messages to the channel in sequence, but it is the responsibility of the channel to deliver these messages to the user; this can be affected by network or server

issues.

Adding responses

To specify a response in the JSON editor, insert the appropriate JSON objects into the `generic` field of the step response. The following example shows output with two responses of different types (text and an image):

```
{
  "generic":[
    {
      "response_type": "text",
      "values": [
        {
          "text_expression": {
            "concat": [
              {
                "scalar": "This is a text response."
              }
            ]
          }
        }
      ]
    },
    {
      "response_type": "image",
      "source": "https://example.com/image.jpg",
      "title": "Example image",
      "description": "This is an image response."
    }
  ]
}
```

For more information, see Response types.

Targeting specific integrations

If you plan to deploy your assistant to multiple channels, you might want to send different responses to different integrations based on the capabilities of each channel. The `channels` property of the generic response object provides a way to do this.

This mechanism is useful if your conversation flow does not change based on the integration in use, and if you cannot know in advance what integration the response is sent to at run time. By using `channels`, you can define a single step that supports all integrations, while still customizing the output for each channel. For example, you might want to customize the text formatting, or even send different response types, based on what the channel supports.

Using `channels` is useful along with the `channel_transfer` response type. Because the message output is processed both by the channel that initiates the `transfer` and by the target channel, you can use `channels` to define responses that are processed by one or the other.

To specify the integrations for which a response is intended, include the optional `channels` array as part of the response object. All response types support the `channels` array. This array contains one or more objects by using the following syntax:

```
{
  "channel": "<channel_name>"
}
```

The value of `<channel_name>` can be any of the following strings:

- `chat`: Web chat
- `voice_telephony`: Phone
- `text_messaging`: SMS
- `slack`: Slack
- `facebook`: Facebook Messenger
- `whatsapp`: WhatsApp

The following example shows step output that contains two responses: one intended for the web chat integration and one intended for the Slack and Facebook integrations.

```
{
```

```
"generic": [
 {
  "response_type": "text",
  "channels": [
   {
    "channel": "chat"
   }
  ],
  "values": [
   {
    "text_expression": {
     "concat": [
      {
       "scalar": "This output is intended for the <strong>web chat</strong>."
      }
     ]
    }
   }
  ]
 },
 {
  "response_type": "text",
  "channels": [
   {
    "channel": "slack"
   },
   {
    "channel": "facebook"
   }
  ],
  "values": [
   {
    "text_expression": {
     "concat": [
      {
       "scalar": "This output is intended for either Slack or Facebook."
      }
     ]
    }
   }
  ]
 }
]
}
```


If the `channels` array is present, it must contain at least one channel object. Any integration that is not listed ignores the response. If the `channels` array is absent, all integrations handle the response.

Response types

You can configure different types of responses using JSON. To learn more about response types and supported integrations for JSON response types, see Response types reference.

Dynamic options

An *options* response presents customers with a list of choices to select from. You can use the **dynamic** setting to generate the list from options that might be different each time.

Dynamic options are generated based on the data stored in a variable, which must be available to the step asking the question. The source variable must contain an array of values, each of which represents one of the options that will be presented to the customer. The items in the array can be simple values such as strings or numbers (for example, `[ "Raleigh", "Boston", "New York" ]`) or compound JSON objects.

A common scenario for dynamic options is when an array is returned from an external API that you call using a custom extension. For example, you might use a custom extension to retrieve a list of credit cards associated with a customer's account. You can then use dynamic options to ask the customer which card to use during the conversation. (For more information about custom extensions, see Calling a custom extension.)

Your actions might also populate the source variable using expressions. For example, you might use a session variable to build a shopping cart containing items the customer has decided to purchase. An action for removing an item from the cart could then use dynamic options to show the items in the cart so the customer can select which one to remove. (For more information about using expressions for variable values, see Using an expression to assign a value

to a session variable.)

Defining dynamic options

To define a dynamic options customer response:

1. In a step, click **Define customer response**.
2. Choose the **Options** response type.
3. Click the **Dynamic** toggle.
4. In the **Source variable** field, choose the variable that contains the array that defines the dynamic options (for example, the variable containing the response from a custom extension that you called in a previous step).
5. **Optional:** In the **Option** field, write an expression that maps the items in the source array to the options that will be listed. This expression serves as a template that converts each item in the array to a meaningful value that will be displayed to the customer. In this expression, use the dynamic variable `${item}` to represent the item.

   In some situations, you do not need to specify an expression:

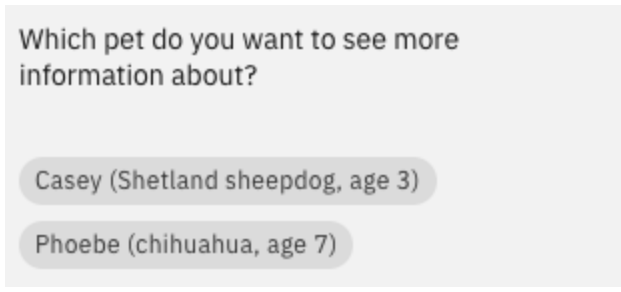
   - If the items in the array are simple values such as strings or integers, the value of each item is automatically shown as an option. However, you might still want to define a mapping if you want to manipulate or reformat the items to make them more meaningful. For example, you might use the expression `"Part #" + ${item}` to show part numbers using the format `Part #12345`.
   - If the items in the array are JSON objects, the default mapping looks for a property called `label` and uses its value (if present) as the option. If the item does not include a `label` property, or you do not want to use the value of the `label` property as the option, you must write an expression to specify a mapping. You can use dot notation to refer to a property in the object using its JSON path (for example, `${item}.name`).
6. **Optional:** Click **Add fallback option** to include a static choice, such as `None of the above`, if the options aren't what the customer wants. You can then add a step that is conditioned on this static option to provide further assistance. To add the condition, write a expression such as `${step_id}.value == "None of the above"`.

Mapping examples

Suppose you want to build an action that shows a list of pets available for adoption and prompts the customer to select a pet to see more information about. The source variable contains an array from a custom extension in the following format:

```
$ [
  {
    "id": "123",
    "name": "Casey",
    "breed": "Shetland sheepdog",
    "age": 3
  },
  {
    "id": "987",
    "name": "Phoebe",
    "breed": "chihuahua",
    "age": 7
  }
]
```

The schema for the items does not include a `label` property, so the default mapping is not available. Instead, you might use an expression to build a complex label that includes data taken from several different properties. For example, you might use the expression `${item}.name + " (" + ${item}.breed + ", age " + ${item}.age + ")"` to define the option labels:



Remember that you can use expression methods to manipulate values from the source variable in various ways. For example, you might have an action customers use to select a credit card for payment, but for security reasons you don't want to show the entire card number. You could write an expression that uses the `substring()` method to include only the last four digits of each card number (for example, `"Card ending in " + ${item}.card_number.substring(16, 20)`).

Referencing the selected item

After the customer has selected one of the dynamically generated options, you will probably need to reference the selected item in a subsequent step.

If you reference the action variable representing the customer response, the default is to use the value of the selected option. However, in some situations, you might not want to use the same value that was used to display the option to the customer. Instead, you might need to use a unique identifier or other property that unambiguously identifies the selected option.

For example, if the customer selects a pet to show more information about, you probably need to use a unique identifier (the `id` property in our example) to query the database, since the pet's name, age, and breed might not be unique. Or if the customer is selecting a credit card from options that show only the last four digits, you will need to use the full credit card number to access the account details or complete a transaction.

In this situation, you can write an expression to access the original properties of the selected item:

1. Create or edit a step that comes after the step in which the customer selects from the dynamic options.
2. In the **Variable values** section, write an expression to assign a value to a session variable. (For more information, see Using an expression to assign a value to a session variable.)
3. In the expression editor, type a dollar sign (`$`) and then select the step in which the customer selected the dynamic option.
4. Use the property name `item` to represent the selected item, and dot notation to access its properties. For example, the following expression accesses the `id` property of the item selected in a previous step:

   ```
   $ ${step_331}.item.id
   ```

   You can use a complex expression to construct a value using multiple properties of the selected item. For example, you might use an expression such as `${step_123}.item.firstname + " " + ${step_123}.item.lastname` to construct a person's full name. Use the expression to define the value in whatever format you need to complete any required action.

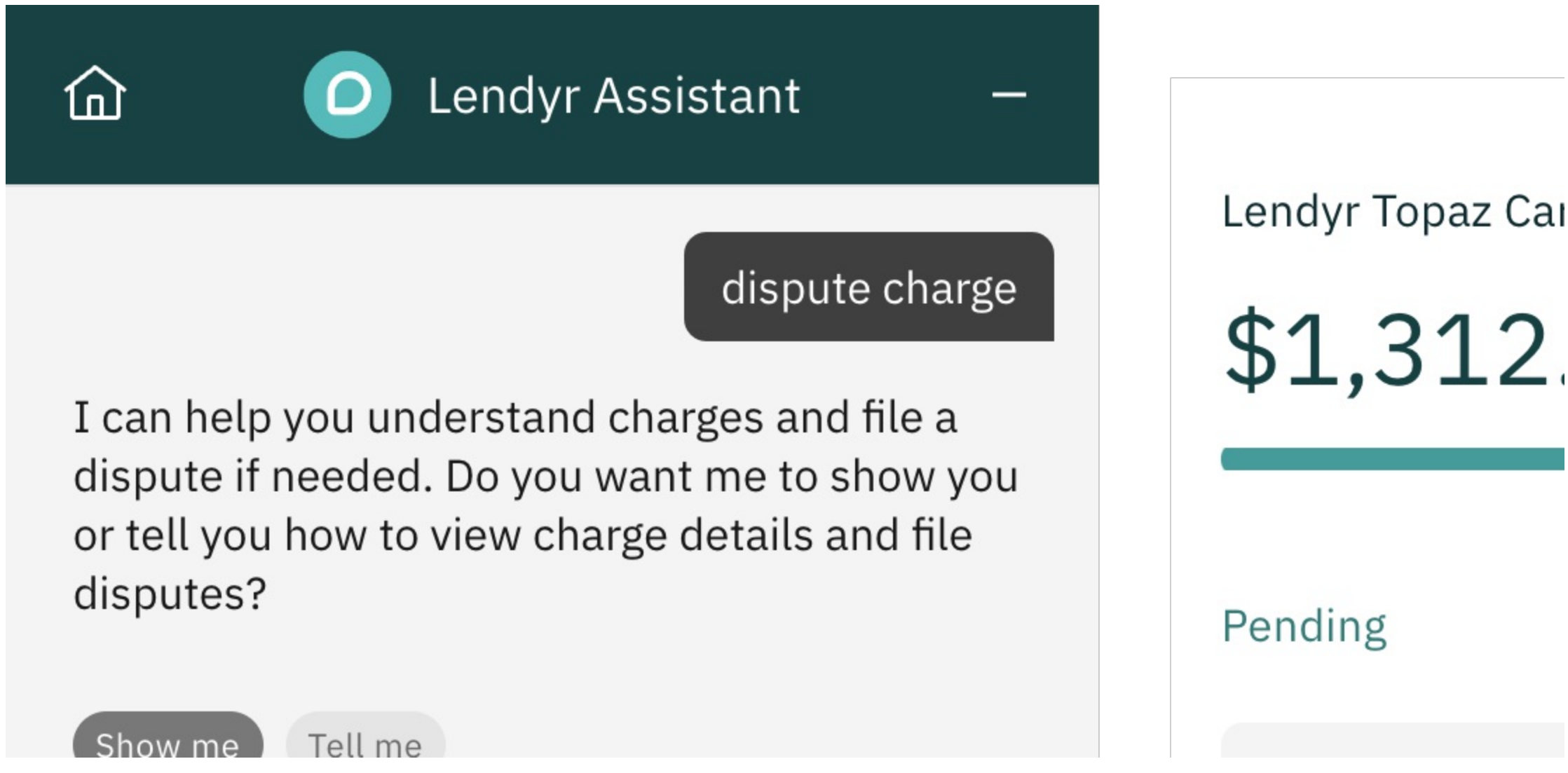
Guiding customers with journeys

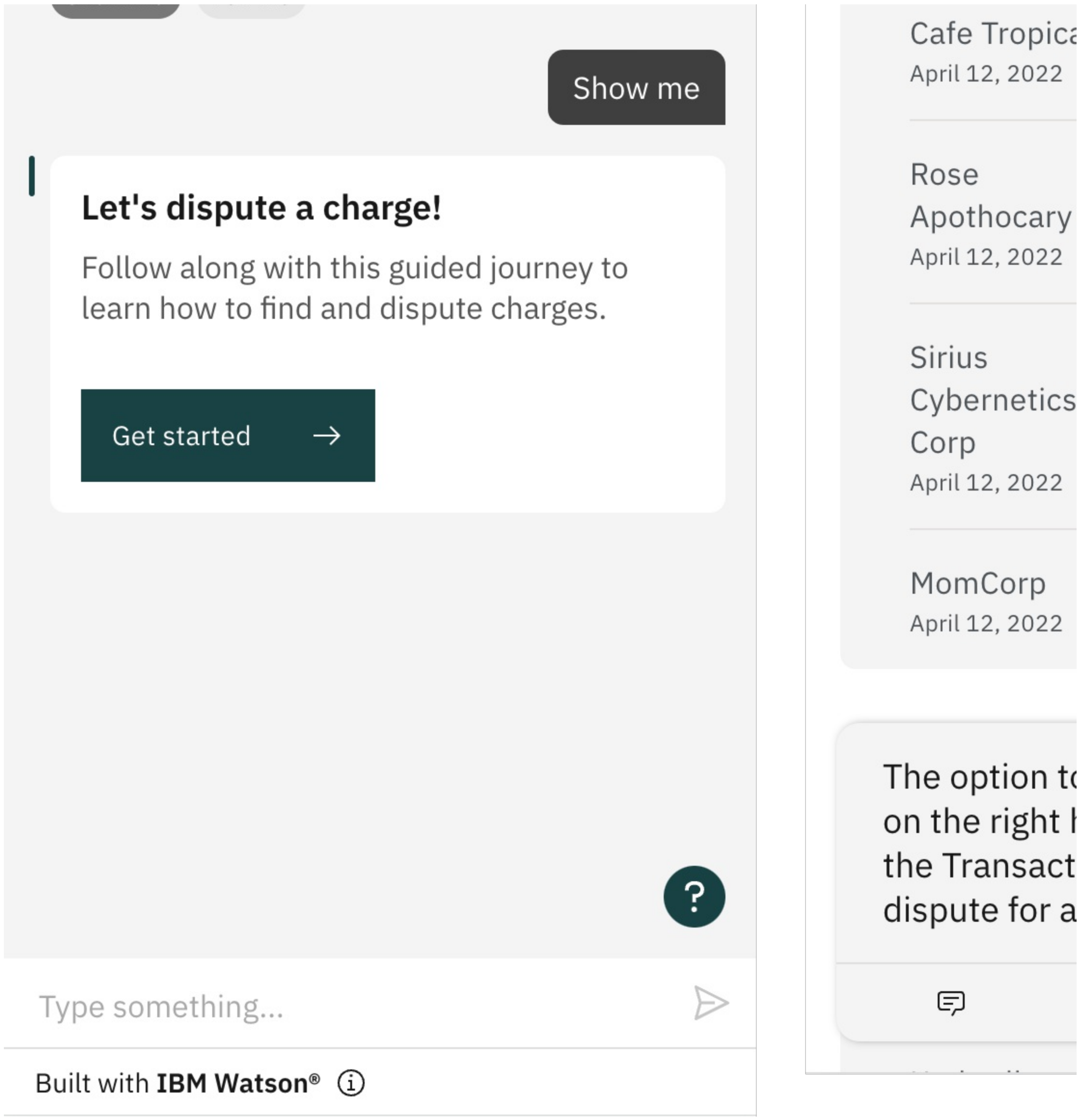
Beta

A journey is an interactive response that you can use to guide your customers through a complex task, or to give them a tour of new features, taking advantage of capabilities your website already supports. A journey is a multipart response that can combine text, video, and images presented in sequence.

This beta feature is available for evaluation and testing purposes only. Journeys require web chat version 6.9.0 or later.

When the customer starts a journey, the chat window temporarily closes. The web chat integration then presents the journey elements one step at a time in a small window superimposed over your website, enabling your customers to navigate and use the website as they step through the journey. At any time during the journey, the customer can freely return to the assistant chat window and then resume the journey.





You might use a journey in situations like the following examples:

- Onboarding new customers to your product or website and showing them where everything is
- Providing customers with step-by-step guidance for a complex task, like filing a claim or creating an account
- Promoting sales opportunities in your product to target users during specific marketing opportunities, such as offering a new rewards program to customers who are concerned about expenses

Tip: For more information about deciding when and how to use journeys, see our best practices guide.

Creating a journey

A journey is defined using the `user_defined` response type, which is available only in the JSON editor. (For more information, see Defining responses using

the JSON editor.) To create a journey, follow these steps:

1. In the action editor, create or edit the step from which you want to start the journey.
2. Click the **Switch to JSON editor** icon </> to open the JSON editor.
3. In the `generic` array, create a `user_defined` response. (For more information, see Defining responses using the JSON editor .)

   A journey is defined using the following structure:

```
$ "user_defined": {
  "user_defined_type": "IBM_BETA_JOURNEYS_TOUR",
  "skip_card": true|false,
  "card_title": "{title}",
  "card_description": "{description}",
  "steps": [
    ...
  ]
}
```

   where:

   `user_defined_type`

   The specific type of user-defined response you are defining. To define a journey, always set this property to `IBM_BETA_JOURNEYS_TOUR`.

   `skip_card`

   An optional property that specifies whether the web chat should start the journey immediately without waiting for the customer to click the introductory card in the web chat window. (The default value is `false`.)

   > **Tip:** You can use this option to start a journey directly from your website, even if the web chat is not open. For more information, see Starting a journey without opening the web chat .

   `card_title`

   The title to display on the introductory card that appears in the web chat when a journey is available (for example, `Website tour` or `Disputing a charge`).

   `card_description`

   The description to display on the introductory card. Describe the journey so your customers can decide whether they want to open it.

   `steps`

   An array of responses defining the steps in the journey.

Defining steps

Each step in a journey is defined as a JSON object describing a response to be shown to the customer, using a format that is similar to how you define assistant responses directly in the `generic` array. Steps in a journey are shown to the customer one at a time, in the order in which you list them in the `steps` array.

As with assistant responses, the `response_type` property identifies the type of response:

`text`

A step that shows only text.

```
$ {
  "response_type": "text",
  "text": "This is the text of the response."
}
```

Markdown formatting and links are supported in `text` steps. For more information, see Markdown formatting.

> ⚠ **Important:** Note that the structure of a `text` step in a journey is different from the `text` response type for assistant responses. Instead of an array of text values, only a single `text` component is supported.

`image`

A step that shows an image, along with an optional description.

```
$ {
  "response_type": "image",
  "source": "https://example.com/image.png",
  "description": "This is the description of the image."
}
```

The `source` property must be the `https:` URL of a publicly accessible image. The specified image must be in `.jpg`, `.gif`, or `.png` format.

`video`

A step that shows a video, along with an optional description.

```
$ {
  "response_type": "video",
  "source": "https://example.com/videos/example-video.mp4",
  "description": "This is the description of the video."
}
```

The URL specified by the `source` property can be either of the following:

- The URL of a video file in a standard format such as MPEG or AVI. In the web chat, the linked video will render as an embedded video player.

  HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.

- The URL of a video hosted on a supported video hosting service. In the web chat, the linked video will render using the embeddable player for the hosting service.

  Specify the URL you would use to view the video in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

  You can embed videos hosted on the following services:

  - YouTube
  - Facebook
  - Vimeo
  - Twitch
  - Streamable
  - Wistia
  - Vidyard

Example

The following example defines a journey that shows users how to dispute a charge, using a combination of text, image, and video responses.

```
$ {
  "generic": [
   {
     "response_type": "user_defined",
     "user_defined": {
       "card_title": "Let’s dispute a charge!",
       "card_description": "Follow along with this guided journey to learn how to find and dispute charges.",
       "user_defined_type": "IBM_BETA_JOURNEYS_TOUR",
       "steps": [
         {
           "response_type": "text",
           "text": "Charges are listed on the Transactions page. **Click your profile photo** in the top right corner of your screen, and then **click Transactions** from the menu."
         },
```

```
        {
          "response_type": "text",
          "text": "Here you can view your charges.\n **Scroll through the Transactions page and review your charges.** Each charge contains a merchant name, transaction date, and amount charged."
        },
        {
          "response_type": "image",
          "source": "https://example.com/image.png",
          "alt_text": "Image showing location of Dispute option",
          "description": "The option to Dispute is marked in red on the right hand side of each row in the Transactions table. Just click here to file a dispute."
        },
        {
          "response_type": "video",
          "source": "https://vimeo.com/769580398",
          "description": "Watch this short video to learn what to expect now that you've filed a dispute."
        }
      ]
    }
  }
 ]
}
```


Starting a journey without opening the web chat

Although journeys are part of the web chat integration, you can make it possible for your customers to start a journey directly from your website without opening the web chat window at all. For example, you might want to include a **Show me** button on your website that customers can click to launch an interactive tour of the page.

To start a journey without opening the web chat:

1. In the action that sends the journey response, edit the JSON that defines the journey. Include `"skip_card": true` to bypass the introductory card.
2. On your website, use the send() instance method to send a message to the assistant that triggers the action that starts the journey (such as `Give me a tour`). Send the message in response to whatever event you want to use to trigger the journey (such as a button click or page load).

Your customers can now start the interactive journey directly from your website without having to open the web chat first. (If the web chat window is opened later, the introductory card for the journey appears in the chat history.)

Limitations

This beta feature currently has the following limitations:

- The preview pane does not support journeys. If you want to preview a journey, use the shareable preview link. For more information about the preview link, see Copying a link to share.
- Journeys use the `view:change` event or `changeView` method, and do not work with the `window:open` or `window:close` events or the `openWindow`, `closeWindow` and `toggleOpen` instance methods.
- When the customer starts a journey, the web chat window closes. If you are using the `view:change` event to trigger the display of a post-chat form, your code should check the value of the new `event.reason` or `event.newViewState.tour` parameter to decide if showing the form is appropriate.

Writing expressions

You can write *expressions* to specify values that are independent of, or derived from, values that are collected in steps or stored in session variables. You can use an expression to define a step condition or to define the value of a session variable.

Using an expression in a step condition

You can use an expression in a step condition if you want to condition a step on the result of a calculation based on information you have gathered during the conversation.

For example, suppose a customer has $200 in a savings account and wants to transfer $150 from it to a new checking account. The funds transfer fee is $3, and the bank charges a fee when a savings account contains less than $50. You could create a step with a step condition that checks for this situation. The step condition would use an expression like this:

```
${savings} - (${Step_232} + ${transfer_fee}) < 50
```

where:

- `${savings}` represents a session variable that stores the customer's savings account total.
- `${Step_232}` represents the step that asks for the amount the customer wants to transfer.
- `${transfer_fee}` represents a session variable that specifies the fee for a funds transfer.

If the step condition is met, the step warns the user that the requested transfer will bring the savings account balance below the $50 minimum and incur a fee, and ask to confirm before proceeding.

To use an expression in a step condition, follow these steps:

1. From the step, click **Add condition**.

   A condition is generated automatically with the most likely choice, which is typically any variables that were set in the previous step.

2. Click the first segment of the generated condition, and then scroll down and click **Expression**.

3. **Optional:** Click the **Expand** icon to open the expression editor window. (You can also type the expression directly in the field without opening the window, but the editor makes it easier to edit a longer or more complex expression.)

4. Type the expression that you want to use.

Using an expression to assign a value to a session variable

You can use an expression when assigning a value to a session variable if you want the variable's value to be calculated based on other variables.

For example, suppose you want to tell your customer the total cost of a purchase, including 6% sales tax and a flat $3.00 processing fee. To calculate the total cost, you could create a session variable and assign the value using an expression:

```
$ (${price} * 1.06) + 3
```

You can then reference this variable in the **Assistant says** field.

To use an expression when assigning a value to a session variable, follow these steps:

1. From within a step, click **Set variable values**.
2. Click **Set new value**.
3. From the drop-down list, select the session variable you want to store the value in.
4. After **to**, select **Expression**.
5. Type the expression you want to use.
6. If you are using the expression editor, click **Apply** to save your changes and close the editor window.

Tip: You can also use an expression to assign an initial value to a session variable. In the **Session variable** window, go to the **Initial value** field and click **Use expression**.

You can also write an expression directly without first picking a variable:

1. From within a step, click **Set variable values**.
2. Click **Set new value**.
3. From the drop-down list, select **Expression**.
4. Type the expression you want to use.
5. If you are using the expression editor, click **Apply** to save your changes and close the editor window.

Expression syntax

The watsonx Assistant expression language is based on the Spring Expression Language (SpEL), but with some important differences in syntax. For detailed background information about SpEL, see Spring Expression Language (SpEL).

Variables

To reference a variable in an expression, type type a dollar sign (`$`) and then select a variable from the list. The reference is inserted into your expression in the correct notation, referencing the variable using its variable ID rather than its display name (for example, `${step_773}` or `${customer_id}`). Do not edit

this reference unless you want to refer to a different variable and you are sure of its variable ID.

To reference the user-defined action or session variables in a custom built client application, use the reserved keyword prefix `user_defined_`. For example, use `${user_defined_my_context_var}` to get the value of `my_context_var` set by your client. For more information on custom built client, see custom built client.

Standard math

For numeric values, you can use expressions to perform mathematical calculations. For basic arithmetic, you can use standard operators (`+`, `-`, `*`, `/`).

You can also use methods to perform additional mathematical operations. For more information, see Expression language methods for actions.

Arrays

To define an array value, type the value using square brackets, with commas separating the items (for example, `[ "one", "two", "three" ]`).

To reference an item in an array, use bracket notation and specify the zero-based index of the item in the array For example, `${Items}[0]` represents the first item in the array `Items`.

Tip: You can also use the array method `get()` to retrieve an item from an array. For more information, see Expression language methods for actions.

JSON objects

Use JSON notation to define compound objects in expressions. For example, the following expression assigns a complex JSON object as the value for a variable:

```
$ {
  "name": {
    "firstname": "John",
    "lastname": "Doe"
  },
  "age": 36
}
```

You can use variables and standard math within JSON to create dynamic objects that are calculated at run time. For example, the following expression defines a JSON object that references variables and calculates an average value:

```
$ {
  "temp_1": ${temp_1},
  "temp_2": ${temp_2},
  "avg_temp": (${temp_1} + ${temp_2}) / 2
}
```

To refer to a child object contained in a JSON value, use dot notation to express the path to the object (for example, `${customer}.name.lastname`).

If you need to refer to a child of an object that might or might not be defined, use the safe navigation operator (`?`). For example, the expression `${customer}.name?.lastname` evaluates to `null` if `customer.name` is `null`. (Without the safe navigation operator, an error would result.)

Methods

Use expression language methods to manipulate values (for example, formatting a string or appending an item to an array). For more information about the supported methods for each data type, see Expression language methods for actions.

Building a custom extension

If you need to integrate your assistant with an external service that has a REST API, you can build a custom extension by importing an OpenAPI document.

After you create a custom extension, you can connect it to an assistant as an integration. In your actions, you can then define steps that interact with the external service by calling the extension.

Tip: The watsonx Assistant extension starter kit repo on GitHub provides files and Advanced Usage tips that you can use to quickly build a working extension. Each starter kit includes a tested OpenAPI definition that you can use to create an extension that accesses a third-party API, along with a downloadable JSON file you can import to create an assistant that accesses the extension.

Overview

OpenAPI (formerly known as Swagger) is an open standard for describing and documenting REST APIs. An OpenAPI document defines the resources and operations that are supported by an API, including request parameters and response data, along with details such as server URLs and authentication methods.

An OpenAPI document describes a REST API in terms of *paths* and *operations*. A path identifies a particular resource that can be accessed by using the API (for example, a hotel reservation or a customer record). An *operation* defines a particular action that can be performed on that resource (such as creating, retrieving, updating, or deleting it).

The OpenAPI document specifies all of the details for each operation, including the HTTP method that is used, request parameters, the data included in the request body, and the structure of the response body.

For more information about the OpenAPI specification, see OpenAPI Specification.

When you create a custom extension, you import an OpenAPI document that describes the REST API of an external service. IBM® watsonx™ Assistant parses the OpenAPI document to identify the operations supported by the external service, along with information about the input parameters and response for each operation and supported authentication methods.

After this processing is complete, the custom extension becomes available as a new integration that you can connect to the assistant. Your assistant can then use the extension to send requests to the external service based on conversations with your customers. Values that are included in the response from the service are then mapped to action variables, which can be accessed by subsequent action steps.

(For more information about connecting a custom extension to an assistant, see Add a custom extension.)

Preparing the API definition

To create a custom extension, you need access to an OpenAPI document that describes the REST API you want to integrate with. Many third-party services publish OpenAPI documents that describe their APIs, which you can download and import. For an API that your company maintains, you can use standard tools to create an OpenAPI document that describes it.

> **Tip:** The SwaggerHub website offers an OpenAPI 3.0 Tutorial, and tools to help you develop and validate your OpenAPI document. You can use the online Swagger editor to convert your OpenAPI document to the correct format and OpenAPI version.

The OpenAPI document must satisfy the following requirements and restrictions:

- The document must conform to the OpenAPI 3.0 specification. If you have an OpenAPI (or Swagger) document that uses an earlier version of the specification, you can use the online Swagger editor to convert it to OpenAPI 3.0.
- The document must be in JSON format (YAML is not supported). If you have a YAML document, you can use the online Swagger editor to convert it to JSON.
- The request body in the JSON script must be presented as an object. For example:

```
{
   name: "Bob",
   hobbies: ["sleeping", "eating", "walking"]
}
```

- The size of the document must not be more than `4 MB` if you have a *Plus* or higher plan of watsonx Assistant. However, if you have an *Enteprise* plan with data isolation, the size of the document must not be more than `8 MB`.
- The `content-type` must be `application/json`.
- To stream from an extension, the response content-type must be `text/event-stream`.
- Each operation must have a clear and concise `summary`. The summary text is used in the UI to describe the operations that are available from an action, so it should be short and meaningful to someone who is building an assistant.
- Relative URLs are currently not supported.
- Only Basic, Bearer, OAuth 2.0, and API key authentication are supported.
- For OAuth 2.0 authentication, Authorization Code, Client Credentials, Password, and custom grant types that starts with `x-` are supported. Note that `x-` is used by the IBM IAM authentication mechanism and by watsonx.
- Schemas that are defined by using `anyOf`, `oneOf`, and `allOf` are currently not supported.

Adhering to the response timeout limit

You must adhere to the response timeout limit, which is 48 seconds, while calling an external API. The timeout value for a custom extension is not configurable.

Building the custom extension

Note: When you configure custom extension with OAuth authentication, watsonx Assistant does not support specification or definition of OAuth scopes. However, watsonx Assistant provides support for OAuth scopes through the newer `OAuth 2.0` authentication used in pre-message and post-message webhooks.

To build a custom extension based on the API definition, follow these steps:

1. Go to the **Integrations** page.
2. Scroll to the **Extensions** section and click **Build custom extension**.
3. Read the **Get started** information and click **Next** to continue.
4. In the **Basic information** step, specify the following information about the extension you are creating:
   - **Extension name**: A short, descriptive name for the extension (for example, `CRM system` or `Weather service`). This name that is displayed on the tile for the extension on the **Integrations** page, and in the list of available extensions in the action editor.
   - **Extension description**: A brief summary of the extension and what it does. The description is available from the **Integrations** page.

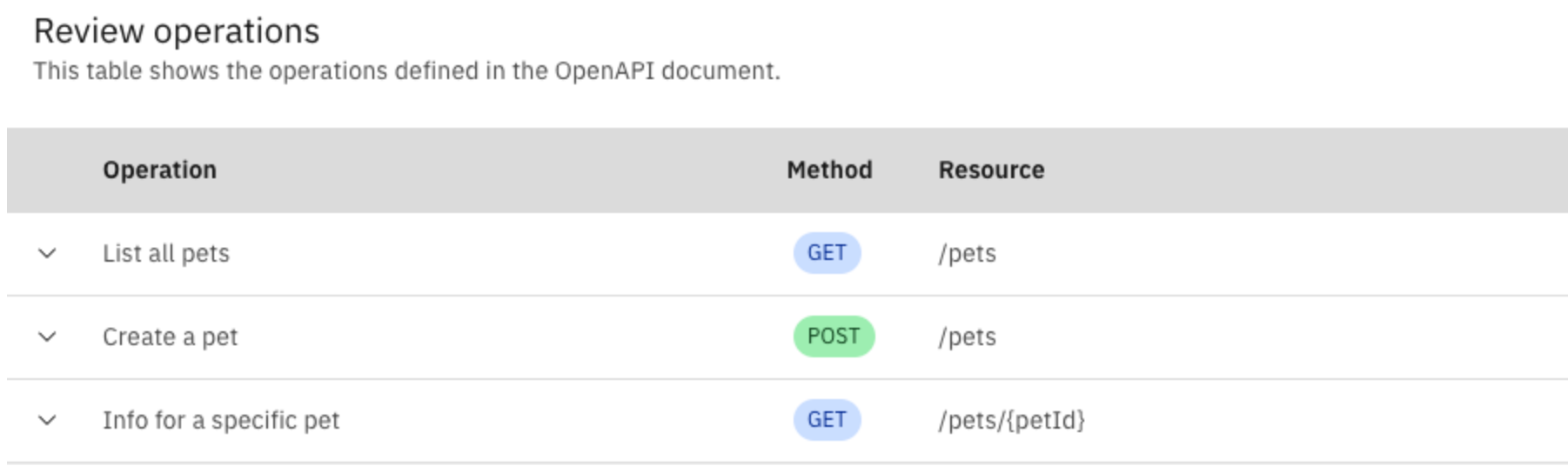
   Click **Next**.
5. In the **Import OpenAPI** step, click or drag to add the OpenAPI document that describes the REST API you want to integrate with.

   If you encounter an error when you try to import the JSON file, make sure the file satisfies all requirements listed in Preparing the API definition. Edit the file to correct errors or remove unsupported features. Click the **X** to clear the error message, and try the import again.

   After you import the file successfully, click **Next**.
6. In the **Manage extension** step, you can review and replace the imported OpenAPI document if required. For more information about replacing the OpenAPI document, see Replacing the OpenAPI document.
7. In the **Authentication** tab, you see information about the authentication methods that are defined in the OpenAPI document. *Table. Fields in Authentication tab* gives details about the fields in the Authentication tab:

Field name	Description	Values
Authentication type	The type of authentication set up in the OpenAPI script.	- `OAuth 2.0` - `Basic Auth` - `API key auth` - `Bearer auth`
Username	The username credential in the OpenAPI script.	For example, `user`
Password	The password credential set up in the OpenAPI script.	For example, `Password@123`
Servers	The link to the server that is defined in the Open API document to connect. to the API extension.	For example, `https://custom-extension-server.xyz`

8. The **Review operations** table shows the operations that the assistant is able to call from an action step. An *operation* is a request by using a particular HTTP method, such as `GET` or `POST`, on a particular resource.



$ For each operation, a row in the table shows the following information:

- **Operation**: A description of the operation, which is derived from either the `summary` (if present) or `description` in the OpenAPI file.
- **Method**: The HTTP method used to send the API request for the operation.
- **Resource**: The path to the resource the operation acts upon.

To see more information about an operation, click the icon next to its row in the table. The following details are shown:

- **Request parameters**: The list of input parameters defined for the operation, along with the type of each parameter and whether the parameter is required or optional.
- **Response properties**: The properties of the response body that are mapped to variables the assistant can access.

9. If you are satisfied with the extension, click **Finish**.

   If you want to change something, delete the extension, edit the JSON file to make your changes, and repeat the import process.

The new extension is now available as a tile in the **Extensions** section of the integrations catalog, and you can add it to your assistant.

Replacing OpenAPI document

To replace an existing OpenAPI document, do the following steps:

1. Go to the **Integrations** () > **Extensions**.
2. Click the **Open** button in the custom extension card for which you want to change the OpenAPI documentation.
3. On the **Open Custom extension** dialog, click **Confirm** to go to the **Manage extension** tab.
4. Click the **Replace** button to select the new OpenAPI document from your system and click **Open**.
5. You can review the operators in the **Review operations** section in the **Manage extension** tab.
6. Review and update the Authentication information in the **Authentication** tab after replacing the OpenAPI document.
7. Go to the **Actions** page and repair any broken action skill because of the replacement of the OpenAPI document.

Adding an extension to your assistant

After you build a custom extension, you must add it to the assistant before it can be accessed by actions.

Adding the extension to the assistant configures the extension for use within a particular environment, and it makes the extension available so that it can be called from actions.

You can use different configuration details for each environment. For example, you might want to use the URL for a test server in the draft environment, but a production server in the live environment.

For information about how to create a custom extension, see Build a custom extension.

Adding the extension to the draft environment

To add a custom extension to the assistant, follow these steps:

1. On the **Integrations** page, scroll to the **Extensions** section and find the tile for the custom extension you want to add.
2. Click **Add**. Review the overview of the extension and click **Confirm** to configure it for your assistant.

> ⚠ **Important:** When you first add an extension to an assistant, the configuration settings you provide are applied only to the draft environment. You must complete configuration for the draft environment before you can add the extension in the live environment.

3. Read the information in the **Get started** step, and then click **Next**.
4. In the **Authentication** step, specify the authentication and server information you want your assistant to use when it calls the service.
   - In the **Authentication type** field, select the type of authentication to use (or **No authentication** if the API is not authenticated). The available authentication types are determined by the security schemes that are defined in the OpenAPI document.
   - Specify the additional information required for the authentication type you selected (such as the username and password, API key, bearer token, or OAuth 2.0 details).

     > ☑ **Tip:** For more information about configuring OAuth 2.0 authentication, see OAuth 2.0 authentication.

   - In the **Servers** field, select the server URL to use.

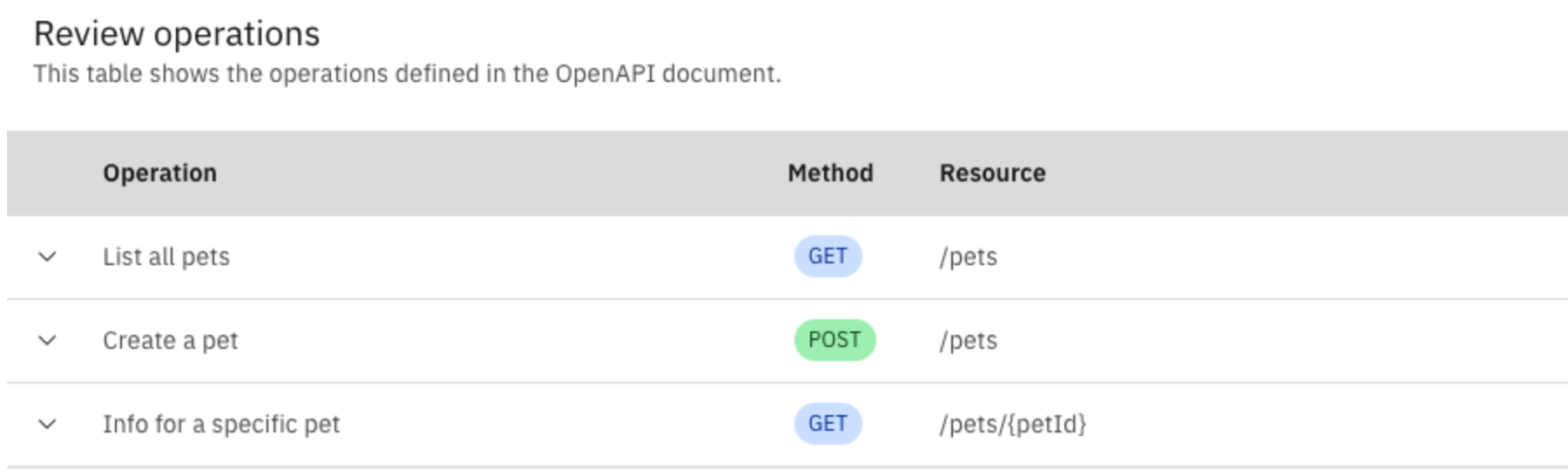
     If the selected URL contains any variables, also specify the values to use. Depending on how each variable is defined in the OpenAPI document, you can either select from a list of valid values or type the value to use in the field.

     The **Generated URL** message shows the full URL that the assistant uses, including the variable values.

   Click **Next**.
5. In the **Manage extension** step, you can review and replace the imported OpenAPI document if required. For more information about replacing the OpenAPI document, see Replacing the OpenAPI document.
6. In the **Authentication** tab, you see information about the authentication methods that are defined in the OpenAPI document. *Table. Fields in Authentication tab* gives details about the fields in the Authentication tab:

Field name	Description	Values
Authentication type	The type of authentication set up in the OpenAPI script.	- `OAuth 2.0` - `Basic Auth` - `API key auth` - `Bearer auth`
Username	The username credential in the OpenAPI script.	For example, `user`
Password	The password credential set up in the OpenAPI script.	For example, `Password@123`
Servers	The link to the server that is defined in the Open API document to connect. to the API extension.	For example, `https://custom-extension-server.xyz`

7. The **Review operations** table shows the operations that the assistant is able to call from an action step. An *operation* is a request by using a particular HTTP method, such as `GET` or `POST`, on a particular resource.



Review operations table

For each operation, a row in the table shows the following information:

- **Operation**: A description of the operation, which is derived from either the `summary` (if present) or `description` in the OpenAPI file.

- **Method**: The HTTP method used to send the API request for the operation.
- **Resource**: The path to the resource the operation acts upon.

To see more information about an operation, click the icon next to its row in the table. The following details are shown:

- **Request parameters**: The list of input parameters defined for the operation, along with the type of each parameter and whether the parameter is required or optional.
- **Response properties**: The properties of the response body that are mapped to variables the assistant can access.

8. Click **Finish**.
9. Click **Close** to return to the Integrations page.

The extension is now connected to your assistant and available for use by actions in the draft environment.

OAuth 2.0 authentication

If you are configuring OAuth 2.0 authentication, the information you must provide depends upon the grant type.

Tip: For more information about OAuth 2.0, see OAuth 2.0.

To complete the OAuth authentication setup, follow these steps:

1. If you haven't already, register your application with the external API you want to access. Copy the client ID and client secret that is provided by the external API.
2. In the **Grant type** field, select the grant type that you want to use. Available grant types are determined by flows that are defined in the `securitySchemes` object in the OpenAPI document. Authorization Code, Client Credentials, Password, and custom grant types that start with `x-` are supported.

   Note: The OAuth2 custom grant type `x-<any custom name>` is used by the IBM IAM authentication mechanism and by watsonx.

3. Specify the required values that were provided by the external API when you registered your application. The required values depend on the grant type:

Grant type	Required values
Authorization Code	• **Client ID** • **Client secret**
Client Credentials	• **Client ID** • **Client secret**
Password	• **Client ID** • **Client secret** • **Username** • **Password**
x-<any custom name>	• **A list of secret fields mentioned in the openAPI spec file**

Grant types

4. If you are using the Authorization Code grant type, follow these steps:
   a. Copy the redirect URL from the watsonx Assistant extension settings page and paste it into the appropriate field on the application registration page for the external API. (The redirect URL is sometimes called the *callback URL*.)
   b. Click **Grant Access**. You are redirected to the authorization page on the website for the external service. Verify that the correct access is being granted and click to approve. You are then redirected back to the extension setup page by using the redirect URL.
5. In the **Client authentication** field, specify whether the authentication credentials are sent in an HTTP header or as part of the request body.

(Credentials that are sent in the request body use the `x-www-form-urlencoded` content type.) Select the option that is expected by the external service.

6. In the **Header prefix** field, specify the prefix that precedes the access token in the `Authorization` header. (The default prefix is `Bearer`, which is typical for most applications.)
7. If you are using the custom grant type `x-<any custom name>` (for example, x-apikey), follow these steps:
    a. Add the secret values associated with the secret fields.
    b. Add the optional parameter values, if any.

Note: If the external service supports the Refresh Token grant type, watsonx Assistant automatically obtains a new access token when the old one expires. If the OpenAPI document defines the `refreshUrl` attribute, the specified URL is used; otherwise, the `tokenUrl` URL is used.

Configuring the extension for the live environment

To configure the extension for the live environment, follow these steps:

1. On the **Integrations** page, scroll to the **Extensions** section and find the tile for the custom extension you want to add.
2. Click **Open**. The **Open custom extension** window opens.
3. In the **Environment** field, select **Live**. Click **Confirm**.
4. Repeat the configuration process, specifying the values that you want to use for the live environment.

Note: If you are using multiple environments, follow the same steps to configure the extension for each environment. For more information, see Adding and using multiple environments.

The extension is now available in the environments that you configured, and it can be called from the assistant. For more information about how to call an extension from an action, see Calling a custom extension.

Plan limits

The number of custom extensions you can add to an assistant depends on your plan. If you already added the maximum number of extensions to the assistant, you must remove one before you can add a new one.

Plan	Extensions per assistant
Enterprise	100
Premium (legacy)	100
Plus	10
Trial	5
Lite	3
Standard (legacy)	0

Extension limits by plan

Streaming from an extension

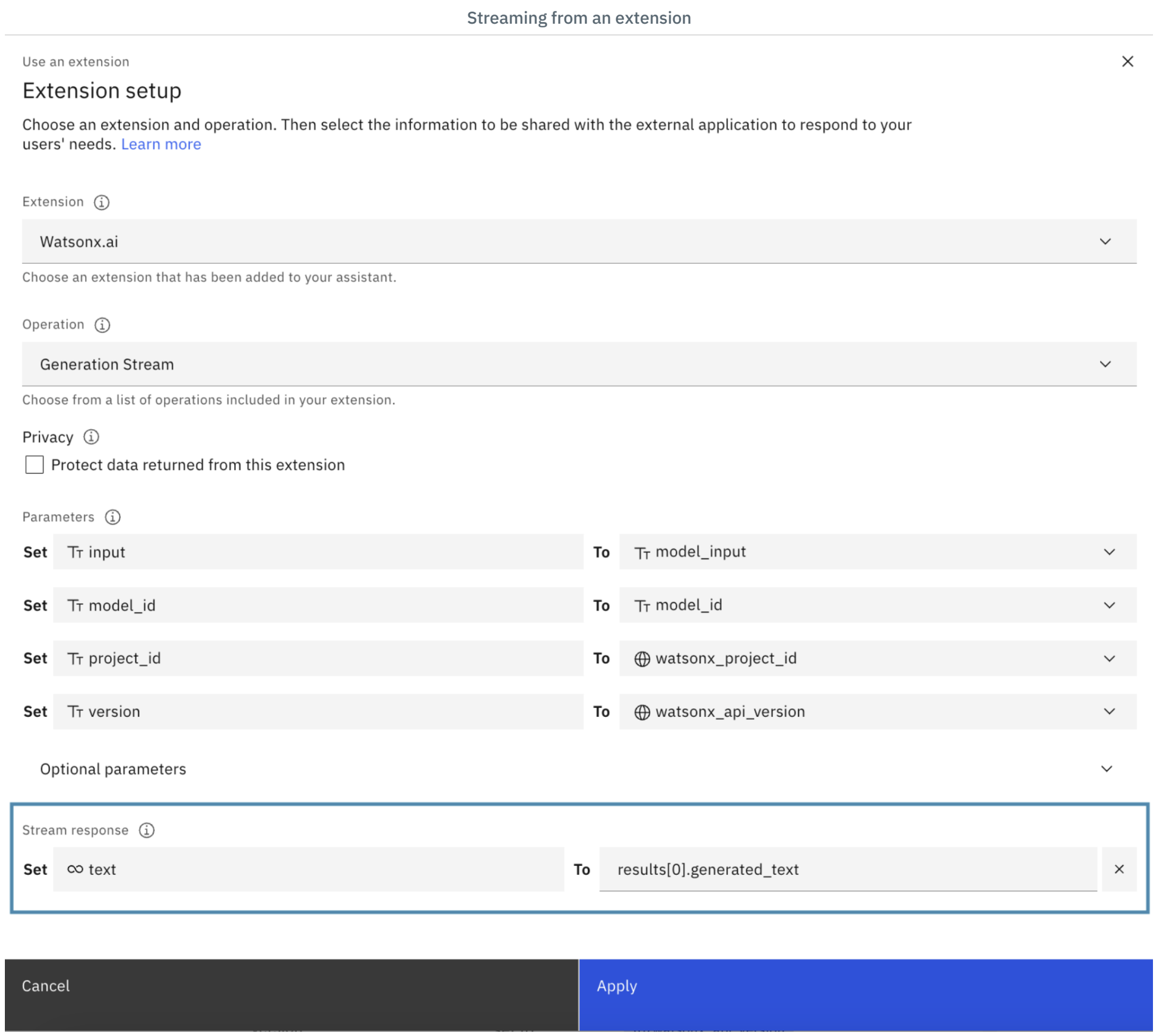
Before you begin

If you want to enable the streaming on webchat, do the following steps:

- Go to **Home** > **Preview** > **Customize web chat**.
- Click the **Styles** tab.
- Set the **Streaming** toggle button to `On`.
- Click **Save and exit**.

How to stream from an extension

You can follow the steps that are outlined in Call the custom extension. If the selected **Operation** defines `text/event-stream` as the response `content-type`, then the feature enables the selection of text to be streamed from SSE generated by the operation.



If your operation generates Server-Sent Events like the following examples:

```
$ data: {"model_id":"ibm/granite-13b-chat-v2","created_at":"2024-07-01T21:49:29.696Z","results":[{"generated_text":"to
lear","generated_token_count":10,"input_token_count":0,"stop_reason":"not_finished"}]}

data: {"model_id":"ibm/granite-13b-chat-v2","created_at":"2024-07-01T21:49:29.731Z","results":[{"generated_text":"n
ne","generated_token_count":11,"input_token_count":0,"stop_reason":"not_finished"}]}

data: {"model_id":"ibm/granite-13b-chat-v2","created_at":"2024-07-01T21:49:29.767Z","results":[{"generated_text":"w
skill","generated_token_count":12,"input_token_count":0,"stop_reason":"not_finished"}]}
```

Specify `text` as the JSON Path in watsonx Assistant to indicate where the text or token is extracted from each SSE. In the previous example, `results[0].generated_text` is the path from where the text is extracted, resulting in: `to learn new`.

Running the stream

When the custom extension is called, watsonx Assistant attempts a call to stream text from the specified **Operation**. The next step runs only after the stream is completed successfully or whether an error occurs before the streaming starts. If the stream completes successfully, the streamed text is stored in the `.body` property of the result variable.

Note: You can access properties from the last event in the stream by referencing the `.last_event` property. For example, use `${step_596_result_1.last_event.citations}` to retrieve specific data.

Checking success or failure

Use the `Ran successfully` response variable to check whether the stream completes successfully or to handle cases where the stream fails to start. This variable returns a boolean value (true or false). It is `true` if the stream completes successfully and `false` if the stream fails to start. For details on conditioning based on HTTP status, see Conditioning on HTTP status to check how to condition.

Note: If the stream fails after it starts due to network issues or it reaches the maximum 30-second timeout, an error will be generated for the user.

Building a custom client

watsonx Assistant API overview

You can use the watsonx Assistant REST APIs, and the corresponding SDKs, to develop applications that interact with the service.

Client applications

To build a virtual assistant or other client application that communicates with an assistant at run time, use the v2 API. You can develop a customer-facing client that can be deployed for production use, an application that brokers communication between an assistant and another service, or a testing application.

By using the v2 runtime API to communicate with your assistant, your application can take advantage of the following features:

- **Automatic state management.** The v2 runtime API manages each session with a user, storing and maintaining all of the context data your assistant needs for a complete conversation.
- **Ease of deploying assistants.** In addition to supporting custom clients, an assistant can be easily deployed to popular messaging channels such as Slack and Facebook Messenger.
- **Versioning.** You can save a snapshot of your content and link your assistant to that specific version. You can then continue to update your development version without affecting the production assistant.
- **Search capabilities.** The v2 runtime API can be used to receive responses from the search integration. When a query is submitted that your assistant can't answer, the search integration can find the best answer from the configured data sources.

For more information, see the watsonx Assistant v2 API reference.

Note: The watsonx Assistant v1 API supports the `/message` method that sends user input directly to the workspace used by a dialog. The v1 runtime API is supported primarily for compatibility. If you use the v1 `/message` method, you must implement your own state management, and you cannot take advantage of versioning or any of the other features of an assistant.

Authoring applications

The v1 API provides methods that enable an application to create or modify dialog skills, as an alternative to building a skill graphically by using the classic Watson Assistant user interface. An authoring application uses the API to create and modify skills, intents, entities, dialog nodes, and other artifacts that make up a dialog skill. For more information, see the v1 API Reference.

Note: The v1 authoring methods create and modify workspaces rather than skills. A workspace is a container for the dialog and training data (such as intents and entities) within a dialog skill. If you create a new workspace by using the API, it appears as a new dialog skill in the classic Watson Assistant user interface.

For a list of the available API methods, see API methods summary.

API methods summary

The following tables list the methods that are available with the watsonx Assistant APIs.

Runtime methods

Method	Description	Version	Reference
Create a session	Create a new session. A session is used to send user input to a skill and receive responses. It also maintains the state of the conversation.	v2	cURL node Java python

Delete session	Deletes a session explicitly before it times out.	v2	cURL node Java python
Send user input to assistant (stateful)	Send user input to an assistant and receive a response, with conversation state (including context data) stored by watsonx Assistant during the session.	v2	cURL node Java python
Send user input to assistant (stateless)	Send user input to an assistant and receive a response, with conversation state (including context data) managed by your application.	v2	cURL node Java python

Authoring methods

Method	Description	Version	Reference
List workspaces	List the workspaces associated with a watsonx Assistant service instance.	v1	cURL node Java python
Create workspace	Create a workspace based on component objects. You must provide workspace components that define the content of the new workspace.	v1	cURL node Java python
Get information about a workspace	Get information about a workspace, optionally including all workspace content.	v1	cURL node Java python
Update workspace	Update an existing workspace with new or modified data. You must provide component objects that define the content of the updated workspace.	v1	cURL node Java python
Delete workspace	Delete a workspace from the service instance.	v1	cURL node Java python
List intents	List the intents for a workspace.	v1	cURL node Java python
Create intent	Create a new intent.	v1	cURL node Java python
Get intent	Get information about an intent, optionally including all intent content.	v1	cURL node Java python
Update intent	Update an existing intent with new or modified data. You must provide component objects that define the content of the updated intent.	v1	cURL node Java python

Delete intent	Delete an intent from a workspace.	v1	cURL node Java python
List user input examples	List the user input examples for an intent, optionally including contextual entity mentions.	v1	cURL node Java python
Create user input example	Add a new user input example to an intent.	v1	cURL node Java python
Get user input example	Get information about a user input example.	v1	cURL node Java python
Update user input example	Update the text of a user input example.	v1	cURL node Java python
Delete user input example	Delete a user input example from an intent.	v1	cURL node Java python
List counterexamples	List the counterexamples for a workspace. Counterexamples are examples that are marked as irrelevant input.	v1	cURL node Java python
Create counterexample	Add a new counterexample to a workspace. Counterexamples are examples that are marked as irrelevant input.	v1	cURL node Java python
Get counterexample	Get information about a counterexample. Counterexamples are examples that are marked as irrelevant input.	v1	cURL node Java python
Update counterexample	Update the text of a counterexample. Counterexamples are examples that are marked as irrelevant input.	v1	cURL node Java python
Delete counterexample	Delete a counterexample from a workspace. Counterexamples are examples that are marked as irrelevant input.	v1	cURL node Java python
List entities	List the entities for a workspace.	v1	cURL node Java python
Create entity	Create a new entity, or enable a system entity.	v1	cURL node Java python

Get entity	Get information about an entity, optionally including all entity content.	v1	cURL node Java python
Update entity	Update an existing entity with new or modified data. You must provide component objects that define the content of the updated entity.	v1	cURL node Java python
Delete entity	Delete an entity from a workspace, or disable a system entity.	v1	cURL node Java python
List entity mentions	List mentions for a contextual entity. An entity mention is an occurrence of a contextual entity in the context of an intent user input example.	v1	cURL node Java python
List entity values	List the values for an entity.	v1	cURL node Java python
Add entity value	Create a new value for an entity.	v1	cURL node Java python
Get entity value	Get information about an entity value.	v1	cURL node Java python
Update entity value	Update an existing entity value with new or modified data. You must provide component objects that define the content of the updated entity value.	v1	cURL node Java python
Delete entity value	Delete a value from an entity.	v1	cURL node Java python
List entity value synonyms	List the synonyms for an entity value.	v1	cURL node Java python
Add entity value synonym	Add a new synonym to an entity value.	v1	cURL node Java python
Get entity value synonym	Get information about a synonym of an entity value.	v1	cURL node Java python
Update entity value synonym	Update an existing entity value synonym with new text.	v1	cURL node Java python

Delete entity value synonym	Delete a synonym from an entity value.	v1	cURL node Java python
List dialog nodes	List the dialog nodes for a workspace.	v1	cURL node Java python
Create dialog node	Create a new dialog node.	v1	cURL node Java python
Get dialog node	Get information about a dialog node.	v1	cURL node Java python
Update dialog node	Update an existing dialog node with new or modified data.	v1	cURL node Java python
Delete dialog node	Delete a dialog node from a workspace.	v1	cURL node Java python
List log events in a workspace	List the events from the log of a specific workspace.	v1	cURL node Java python
List log events in all workspaces	List the events from the logs of all workspaces in the service instance.	v1	cURL node Java python
Delete labeled data	Deletes all data that is associated with a specified customer ID. The method has no effect if no data is associated with the customer ID.	v1	cURL node Java python

Building a custom client by using the API

Autolearning feature in watsonx Assistant
Effective June 16, 2025, the autolearning feature is discontinued for watsonx Assistant. After this date, autolearning settings are removed from the **Actions global settings** page, and all autolearning capabilities are disabled.

If none of the built-in integrations meet your requirements, you can deploy your assistant by developing a custom client application that interacts with your users and communicates with the IBM® watsonx™ Assistant service.

The Watson SDKs help you write code that interacts with watsonx Assistant. For more information about the SDKs, see IBM Watson APIs.

Setting up the assistant

The example application that we create in implements several simple functions to illustrate how a client application interacts with watsonx Assistant. The application code collects input and sends it to an assistant, which sends responses the application shows to the user.

To try this example yourself, you first need to set up the simple example assistant that the client connects to:

1. Download the actions JSON file.
2. Create an assistant.

3. In the new assistant, open the global action settings. Go to the **Upload/Download** tab and import the actions from the file you downloaded.

The example actions include a *Greet customer* action that asks the customer's name, and simple actions for making and canceling appointments.

Getting service information

To access the watsonx Assistant REST APIs, your application needs to be able to authenticate with IBM Cloud® and connect to the assistant in the environment where it is deployed. You need to copy the service credentials and environment ID and paste them into your application code. You also need the URL for the location of your service instance (for example, `https://api.us-south.assistant.watson.cloud.ibm.com`).

To find this information:

1. Go to the **Environments** page and choose he environment you want to connect to.
2. Click the **Settings** icon icon to open the environment settings.
3. Select **API details** to see details for the environment, including the service instance URL and environment ID. To find the API key, follow the link in the **Service credentials** section.

Communicating with the watsonx Assistant service

Interacting with the watsonx Assistant service from your client application is simple. We start with an example that connects to the service, sends a single empty message, and prints the output to the console:

Node

```
// Example 1: Creates service object, sends initial message, and
// receives response.

const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};
sendMessage(messageInput);

// Send message to assistant.
function sendMessage(messageInput) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the result.
function processResult(result) {
  // Print responses from actions, if any. Supports only text responses.
  if (result.output.generic) {
    if (result.output.generic.length > 0) {
```

```
    result.output.generic.forEach( response => {
      if (response.response_type == 'text') {
        console.log(response.text);
      }
    });
  }
 }
}
```

Python

```
# Example 1: Creates service object, sends initial message, and
# receives response.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
    version = '2021-11-27',
    authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Start conversation with empty message.
result = assistant.message_stateless(
    assistant_id,
).get_result()

# Print responses from actions, if any. Supports only text responses.
if result['output']['generic']:
    for response in result['output']['generic']:
        if response['response_type'] == 'text':
            print(response['text'])
```

The first step is to create a service object, a sort of wrapper for the watsonx Assistant service.

You use the service object for sending input to, and receiving output from, the service. When you create the service object, you specify the API key for authentication, and the version of the watsonx Assistant API you are using.

In this Node.js example, the service object is an instance of `AssistantV2`, stored in the variable `assistant`. The Watson SDKs for other languages provide equivalent mechanisms for instantiating a service object.

In this Python example, the service object is an instance of `watson_developer_cloud.AssistantV2`, stored in the variable `assistant`. The Watson SDKs for other languages provide equivalent mechanisms for instantiating a service object.

After you create the service object, we use it to send a message to the assistant, by using the stateless `message` method. In this example, the message is empty; we want to trigger the *Greet customer* action to start the conversation, so we don't need any input text. We then print any text responses that are returned in the `generic` array in the returned output.

Use the `node <filename.js>` command to run the example application.

Use the `python3 <filename.py>` command to run the example application.

Note: Make sure you install the Watson SDK for Node.js by using `npm install ibm-watson`.

Note: Make sure you install the Watson SDK for Python by using `pip install --upgrade ibm-watson` or `easy_install --upgrade ibm-watson`.

Assuming everything works as expected, the assistant returns the output from the assistant, which the app then prints to the console:

```
Welcome to the watsonx Assistant example. What's your name?
```

This output tells us that we communicated with the assistant and received the greeting message that is specified by the *Greet customer* action. But we don't yet have a way of responding to the assistant's question.

Processing user input

To be able to process user input, we need to add a user interface to our client application. For this example, we keep things simple and use standard input

and output. You can use the Node.js prompt-sync module. (You can install prompt-sync by using `npm install prompt-sync`.) You can use the Python 3 `input` function.

Node

```
// Example 2: Adds user input.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};
sendMessage(messageInput);

// Send message to assistant.
function sendMessage(messageInput) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the result.
function processResult(result) {

  // Print responses from actions, if any. Supports only text responses.
if (result.output.generic) {
  if (result.output.generic.length > 0) {
    result.output.generic.forEach( response => {
      if (response.response_type === 'text') {
        console.log(response.text);
      }
    });
  }
}

// Prompt for the next round of input unless skip_user_input is true.
let newMessageFromUser = '';
if (result.context.global.system.skip_user_input !== true) {
  newMessageFromUser = prompt('>> ');
}

if (newMessageFromUser !== 'quit') {
  newMessageInput = {
    messageType: 'text',
    text: newMessageFromUser,
  }
  sendMessage(newMessageInput);
```

```
  }
}
```

Python

```
# Example 2: Adds user input.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
    version = '2021-11-27',
    authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Initialize with empty value to start the conversation.
message_input = {
    'message_type:': 'text',
    'text': ''
    }

context = None

# Main input/output loop
while message_input['text'] != 'quit':

    # Send message to assistant.
    result = assistant.message_stateless(
        assistant_id,
        input = message_input,
        context=context
    ).get_result()
    context = response['context']

    # Print responses from actions, if any. Supports only text responses.
    if result['output']['generic']:
        for response in result['output']['generic']:
            if response['response_type'] == 'text':
                print(response['text'])

    # Prompt for the next round of input unless skip_user_input is True.
    if not result['context']['global']['system'].get('skip_user_input', False):
        user_input = input('>> ')
        message_input = {
            'text': user_input
    }
```

This version of the application begins the same way as before: sending an empty message to the assistant to start the conversation.

The `processResult()` function displays the text of any responses that are received from the assistant. It then prompts for the next round of user input.

It then displays the text of any responses that are received from the assistant, and it prompts for the next round of user input.

> **Tip:** The example checks for the global context variable `skip_user_input` and prompts for user input only if this variable is not set to `true True`. The `skip_user_input` variable is set by the assistant in some situations where no user input is needed (for example, if the assistant called an external service but is still waiting for the result). It's good practice always to make this check before you prompt for user input.

Because we need a way to end the conversation, the client app is also watching for the literal command `quit` to indicate that the program should exit.

But something still isn't right:

```
Welcome to the watsonx Assistant example. What's your name?
>> Robert
I'm afraid I don't understand. Please rephrase your question.
>> I want to make an appointment.
```

```
What day would you like to come in?
>> Thursday
I'm afraid I don't understand. Please rephrase your question.
>>
```

The assistant is starting out with the correct greeting, but it doesn't understand when you tell it your name. And if you tell it you want to make an appointment, the correct action is triggered; but again, it doesn't understand when you answer the follow-up question.

The reason is because we are using the stateless `message` method, which means that it is the responsibility of our client application to maintain state information for the conversation. Because we are not yet doing anything to maintain the state, the assistant sees every round of user input as the first turn of a new conversation. Because it has no memory of asking a question, it tries to interpret your answer as a new question or request.

Maintaining the state

State information for your conversation is maintained by using the *context*. The context is an object that is passed back and forth between your application and the assistant, storing information that can be preserved and updated as the conversation goes on. Because we are using the stateless `message` method, the assistant does not store the context, so it is the responsibility of our client application to maintain it from one turn of the conversation to the next.

The context includes a session ID for each conversation and a counter that is incremented with each turn of the conversation. The assistant updates the context and returns it with each response. But our previous version of the example did not preserve the context, so these updates were lost, and each round of input appeared to be the start of a new conversation. We can fix that by saving the context and sending it back to the assistant each time.

In addition to maintaining our place in the conversation, the context can contain action variables that store other data you want to pass back and forth between your application and the assistant. For example, you can include persistent data that you want to maintain throughout the conversation (such as a customer's name or account number), or any other data you want to track (such as the contents of a shopping cart or user preferences).

Node

```
// Example 3: Preserves context to maintain state.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Create Assistant service object.
const assistant = new AssistantV2({
  version: '2021-11-27',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  }),
  url: '{url}', // replace with URL
});

const assistantId = '{environment_id}'; // replace with environment ID

// Start conversation with empty message
messageInput = {
  messageType: 'text',
  text: '',
};
context = {};
sendMessage(messageInput);

// Send message to assistant.
function sendMessage(messageInput, context) {
  assistant
    .messageStateless({
      assistantId,
      input: messageInput,
      context: context,
    })
    .then(res => {
      processResult(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}
```

```
// Process the result.
function processResult(result) {

  let context = result.context;

  // Print responses from actions, if any. Supports only text responses.
  if (result.output.generic) {
    if (result.output.generic.length > 0) {
      result.output.generic.forEach( response => {
        if (response.response_type === 'text') {
          console.log(response.text);
        }
      });
    }
  }

  // Prompt for the next round of input unless skip_user_input is true.
  let newMessageFromUser = '';
  if (result.context.global.system.skip_user_input !== true) {
    newMessageFromUser = prompt('>> ');
  }

  if (newMessageFromUser !== 'quit') {
    newMessageInput = {
      messageType: 'text',
      text: newMessageFromUser,
    }
    sendMessage(newMessageInput, context);
  }
}
```

Python

```
# Example 3: Preserves context to maintain state.

from ibm_watson import AssistantV2
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Create Assistant service object.
authenticator = IAMAuthenticator('{apikey}') # replace with API key
assistant = AssistantV2(
    version = '2021-11-27',
    authenticator = authenticator
)
assistant.set_service_url('{url}') # replace with service instance URL
assistant_id = '{environment_id}' # replace with environment ID

# Initialize with empty message to start the conversation.
message_input = {
    'message_type:': 'text',
    'text': ''
    }
context = {}

# Initialize with empty message to start the conversation.
message_input = {
    'message_type:': 'text',
    'text': ''
    }
context = {}

# Main input/output loop
while message_input['text'] != 'quit':

    # Send message to assistant.
    result = assistant.message_stateless(
        assistant_id,
        input = message_input,
        context = context
```

```
).get_result()

context = result['context']

# Print responses from actions, if any. Supports only text responses.
if result['output']['generic']:
    for response in result['output']['generic']:
        if response['response_type'] == 'text':
            print(response['text'])

# Prompt for the next round of input unless skip_user_input is True.
if not result['context']['global']['system'].get('skip_user_input', False):
    user_input = input('>> ')
    message_input = {
        'text': user_input
    }
```

The only change from the previous example is that we are now storing the context that is received from the assistant in a variable that is called `context`, and we're sending it back with the next round of user input:

The only change from the previous example is that we are now storing the context that is received from the assistant in a variable that is called `context`, and we're sending it back with the next round of user input:

Node

```
assistant
  .messageStateless({
    assistantId,
    input: messageInput,
    context: context,
  })
```

Python

```
response = assistant.message_stateless(
    assistant_id,
    input = message_input,
    context = context
).get_result()
```

This ensures that the context is maintained from one turn to the next, so the watsonx Assistant service no longer thinks every turn is the first:

```
Welcome to the watsonx Assistant example. What's your name?
>> Robert
Hi, Robert! How can I help you?
>> I want to make an appointment.
What day would you like to come in?
>> Next Monday
What time works for you?
>> 10 AM
OK, Robert. You have an appointment for 10:00 AM on Sep 12. See you then!
```

Success! The application now uses the watsonx Assistant service to understand natural-language input, and it displays the appropriate responses.

This simple example illustrates how you can build a custom client app to communicate with the assistant. A real-world application would use a more sophisticated user interface, and it might integrate with other applications such as a customer database or other business systems. It would also need to send more data to the assistant, such as a user ID to identify each unique user. But the basic principles of how the application interacts with the watsonx Assistant service would remain the same.

Including clarifying questions

When your assistant finds that more than one action might fulfill a customer's request, it can automatically ask for clarification. For more information, see Asking clarifying questions.

To include clarifying questions in your custom client, you need to:

- Display the clarification suggestion options that are returned from the `message` API
- Call the `message` API in the next round with a payload that corresponds to the suggestion option that a customer chose to answer the clarifying

question. If you don't implement the call, autolearning and using unrecognized requests to get action recommendations don't work correctly.

Each clarification suggestion includes:

- A label that can be displayed to the customer
- A value that specifies the input that is sent to the assistant if the user chooses the corresponding suggestion

To implement clarification suggestions in your application:

1. Use the `value.input` object from the selected suggestion as the next round of message input, rather than building a new input object. The assistant then responds by triggering the action that is associated with the suggestion option to start.
2. Verify that you implemented this correctly by using your custom client by using the Analyze page. Enter an input that triggers a clarification, and then click the **None of the Above** option. When you view the request in Conversations, check that the user request that initiated clarification is marked with **Unrecognized**, which indicates that your client is properly sending the clarification input to your assistant.

Using the v1 runtime API

Using the v2 API is the recommended way to build a runtime client application that communicates with the watsonx Assistant service. However, some older applications might still be using the v1 runtime API, which includes a similar method for sending messages to the workspace within a dialog skill. If your app uses the v1 runtime API, it communicates directly with the workspace, bypassing the skill orchestration and state-management capabilities of the assistant.

For more information about the v1 `/message` method and context, see the v1 API Reference.

Migrating to the v2 API

The watsonx Assistant v2 runtime API, which supports the use of assistants and skills, was introduced in November 2018. This API offers significant advantages over the v1 runtime API, including automatic state management, ease of deployment, skill versioning, and the availability of new features such as the search skill.

The v2 API is available for all users, regardless of service plan, at no additional cost.

> **Note:** The v2 API currently supports only runtime interaction with an existing assistant. Authoring applications that create or modify workspaces should continue to use the v1 API.

Overview

With the v2 API, your client app communicates with an assistant, which is an orchestration layer that offers several capabilities, including automatic state management, skill versioning, and easier deployment.

All communication with an assistant takes place within the context of a *session*, which maintains conversation state throughout the duration of the conversation. State data, including any context variables that are defined by your dialog or client application, are automatically stored by watsonx Assistant, without any action required on the part of your application.

State data persists until you explicitly delete the session, or until the session times out because of inactivity.

> **Note:** If you prefer to manage state yourself, the v2 API also provides a stateless `message` method that functions more like the v1 API. If you use the stateless `message` method, you do not need to explicitly create or delete sessions, and your app is responsible for maintaining context. For more information about the stateless `message` method, see the API Reference.

If you have an existing application that uses the v1 API to send user input directly to a workspace, migrating your app to use the v2 API is a straightforward process.

Assistant ID

The v2 runtime API sends messages to an assistant. On the **Assistant Settings** page, find the assistant ID. Your application uses this ID to communicate with the assistant. The service credentials are the same for both the v1 and v2 APIs.

> **Note:** Currently, there is no API support for retrieving an assistant ID. To find the assistant ID, you must use the watsonx Assistant user interface.

Call the v2 runtime API

After you create an assistant, you can update your client application to use the v2 runtime API instead of the v1 runtime API.

1. Before sending the first message in a conversation, use the v2 **Create a session** method to create a session. Save the returned session ID:

   Node

   ```
   service
   .createSession({
    assistant_id: assistantId,
   })
   .then(res => {
    sessionId = res.session_id;
   })
   ```

   Python

   ```
   session_id = service.create_session(
      assistant_id = assistant_id
   ).get_result()['session_id']
   ```

   Java

   ```
   CreateSessionOptions createSessionOptions = new CreateSessionOptions.Builder(assistantId).build();
   SessionResponse session = service.createSession(createSessionOptions).execute().getResult();
   String sessionId = session.getSessionId();
   ```

2. Use the v2 **Send user input to assistant** method to send user input to the assistant. Instead of specifying the workspace ID as you did with the v1 API, you specify the assistant ID and the session ID:

   Node

   ```
   service
    .message({
      assistant_id: assistantId,
      session_id: sessionId,
      input: messageInput
    })
   ```

   Python

   ```
   response = service.message(
      assistant_id,
      session_id,
      input = message_input
   ).get_result()
   ```

   Java

   ```
   MessageInput input = new MessageInput.Builder().text(inputText).build();
   MessageOptions messageOptions = new MessageOptions.Builder(assistantId, sessionId)
     .input(input)
     .build();
   MessageResponse response = service.message(messageOptions)
     .execute()
     .getResult();
   ```

   The basic message structure has not changed; in particular, the user input is still sent as `input.text`.

3. After a conversation ends, use the v2 **Delete session** method to delete the session.

   Node

   ```
   service
    .deleteSession({
      assistant_id: assistantId,
      session_id: sessionId,
    })
   ```

Python

```
service.delete_session(
    assistant_id = assistant_id,
    session_id = session_id
)
```

Java

```
DeleteSessionOptions deleteSessionOptions = new DeleteSessionOptions.Builder(assistantId, sessionId
  .build();
service.deleteSession(deleteSessionOptions).execute();
```

If you do not explicitly delete the session, it will be automatically deleted after the configured timeout interval. The timeout duration depends on your plan.

To see examples of the v2 APIs in the context of a simple client application, see Building a custom client using the API.

Handle the v2 response format

Your application might need to be updated to handle the v2 runtime response format, depending on which parts of the response your application needs to access:

- Output for all response types (such as `text` and `option`) are still returned in the `output.generic` object. Application code for handling these responses should work without modification.
- Detected intents and entities are now returned as part of the `output` object, rather than at the root of the response JSON.
- The conversation context is now organized into two objects:
  - The **global context** contains system-level context data that is shared by all skills in the assistant.
  - The **skill context** contains any user-defined context variables that are used by your dialog skill.

  However, keep in mind that state data, including conversation context, is now maintained by the assistant, so your application might not need to access the context at all (see Let the assistant maintain state).

Refer to the v2 API Reference for complete documentation of the v2 response format.

Let the assistant maintain state

For most applications, you can now remove any code included for maintaining state. It is no longer necessary to save the context and send it back to watsonx Assistant with each turn of the conversation. The context is automatically maintained by watsonx Assistant and can be accessed by your dialog as before.

Note that with the v2 API, the context is by default not included in responses to the client application. However, your code can still access context variables if necessary:

- You can still send a `context` object as part of the message input. Any context variables that you include are stored as part of the context maintained by watsonx Assistant. (If the context variable you send already exists in the context, the new value overwrites the previously stored value.)

  Make sure the context object that you send conforms to the v2 format. All user-defined context variables that are sent by your application should be part of the skill context; typically, the only global context variable you might need to set is `system.user_id`, which is used by Plus and Premium plans for billing purposes.
- You can still retrieve context variables from either the global or skill context. To have the `context` object included with message responses, use the **return_context** property in the message input options. For more information, see Accessing context data in dialog.

Accessing context data in dialog

The *context* is an object that contains variables that persist throughout a conversation and can be shared by the dialog and the client application. Both the dialog and the client application can read and write context variables.

You can choose whether you want the context to be maintained by your application or by the watsonx Assistant service:

- If you use the stateful v2 `message` API, the context is automatically maintained by the assistant on a per-session basis. Your application must explicitly create a session at the beginning of each conversation. The context is stored by the service as part of the session and is not returned in message responses unless you request it. For more information, see the v2 API Reference.

- If you use the stateless v2 `message` API (or the legacy v1 `message` API) your application is responsible for storing the context after each conversation turn and sending it back to the service with the next message. For a complex application, or an application that needs to store personally identifiable information, you might choose to store the context in a database.

  A session ID is automatically generated at the beginning of the conversation, but no session data is stored by the service. With the stateless `message` API, the context is always included with each message response. For more information, see the v2 API Reference.

Important: One use of the context is to specify a unique user ID for each user who interacts with the assistant. For user-based plans, this ID is used for billing purposes. For more information, see User-based plans explained.

There are two types of context:

- **Global context**: context variables that are shared by all skills that are used by an assistant, including internal system variables that are used to manage conversation flow. The global context includes the user ID and other global values such as the time zone and language of the assistant.
- **Skill-specific context**: context variables specific to a particular skill, including any user-defined variables needed by your application. Currently, only one skill (named `main skill`) is supported.

User-defined context variables that you specify in a dialog node are part of the `user_defined` object within the skill context when accessed by using the API. This structure differs from the `context` structure that appears in the JSON editor in the watsonx Assistant user interface. For example, you might specify the following code in the JSON editor:

```
"context": {
  "my_context_var": "this is the value"
}
```

In the v2 API, you would access this user-defined variable as follows:

```
"context": {
  "skills": {
    "main skill": {
      "user_defined": {
        "my_context_var": "this is the value"
      }
    }
  }
}
```

For more information about how to access context variables by using the API, see the v2 API Reference.

Example

The following example shows a stateful `/message` request that includes both global and skill-specific context variables; it also uses the `options.return_context` property to request that the context is returned with the response. This option is applicable only if you are using the stateful `message` method because the stateless `message` method always returns the context.

Node

```
service
  .message({
    assistant_id: '{assistant_id}',
    session_id: '{session_id}',
    input: {
      message_type: 'text',
      text: 'Hello',
      options: {
        'return_context': true
      }
    },
    context: {
      'global': {
        'system': {
          'user_id': 'my_user_id'
        }
      },
      'skills': {
        'main skill': {
          'user_defined': {
            'account_number': '123456'
```

```
        }
      }
    }
  }
})
.then(res => {
  console.log(JSON.stringify(res, null, 2));
})
.catch(err => {
  console.log(err);
});
```

Python

```
response=service.message(
    assistant_id='{assistant_id}',
    session_id='{session_id}',
    input={
        'message_type': 'text',
        'text': 'Hello',
        'options': {
            'return_context': True
        }
    },
    context={
        'global': {
            'system': {
                'user_id': 'my_user_id'
            }
        },
        'skills': {
            'main skill': {
                'user_defined': {
                    'account_number': '123456'
                }
            }
        }
    }
).get_result()

print(json.dumps(response, indent=2))
```

Java

```
MessageInputOptions inputOptions = new MessageInputOptions.Builder()
  .returnContext(true)
  .build();

MessageInput input = new MessageInput.Builder()
  .messageType("text")
  .text("Hello")
  .options(inputOptions)
  .build();

// create global context with user ID
MessageContextGlobalSystem system = new MessageContextGlobalSystem.Builder()
  .userId("my_user_id")
  .build();
MessageContextGlobal globalContext = new MessageContextGlobal.Builder()
  .system(system)
  .build();

// build user-defined context variables, put in skill-specific context for main skill
Map<String, Object> userDefinedContext = new HashMap<>();
userDefinedContext.put("account_number","123456");
MessageContextSkill mainSkillContext = new MessageContextSkill.Builder()
  .userDefined(userDefinedContext)
  .build();
Map<String, MessageContextSkill> skillsContext = new HashMap<>();
skillsContext.put("main skill", mainSkillContext);
```

```
MessageContext context = new MessageContext.Builder()
  .global(globalContext)
  .skills(skillsContext)
  .build();

MessageOptions options = new MessageOptions.Builder()
  .assistantId("{assistant_id}")
  .sessionId("{session_id}")
  .input(input)
  .context(context)
  .build();

MessageResponse response = service.message(options).execute().getResult();

System.out.println(response);
```

In this example request, the application specifies a value for `user_id` as part of the global context. In addition, it sets one user-defined context variable (`account_number`) as part of the skill-specific context. This context variable can be accessed by dialog nodes as `$account_number`.

You can specify any variable name that you want to use for a user-defined context variable. If the specified variable exists, it is overwritten with the new value; if not, a new variable is added to the context.

The output from this request includes not only the usual output, but also the context, showing that the specified values have been added. If you are using the stateless `message` method, this context data must be stored locally and sent back to the watsonx Assistant service as part of the next message. If you are using the stateful `message` method, this context is stored automatically and persists for the life of the session.

```
{
  "output": {
    "generic": [
      {
        "response_type": "text",
        "text": "Welcome to the watsonx Assistant example!"
      }
    ],
    "intents": [
      {
        "intent": "hello",
        "confidence": 1
      }
    ],
    "entities": []
  },
  "user_id": "my_user_id",
  "context": {
    "global": {
      "system": {
        "turn_count": 1,
        "user_id": "my_user_id"
      }
    },
    "skills": {
      "main skill": {
        "user_defined": {
          "account_number": "123456"
        }
      }
    }
  }
}
```

Restoring conversation state

In some situations, you might want the ability to restore a conversation to a previous state.

You can use the `export` option on stateful `message` requests to specify that you want the context object in the response to include complete session state data. If you specify `true` for this option, the returned skill context includes an encoded `state` property that represents the current conversation state.

If you are using the stateful `message` API, the service stores conversation state data only for the life of the session. However, if you save this context data

(including `state`) and send it back to the service with a subsequent message request, you can restore the conversation to the same state, even if the original session expired or was deleted.

If you are using the stateless `message` API, the `state` property is always included in responses (along with the rest of `context`). Although stateless sessions do not expire, you can still use this state data to reset a conversation to a previous state.

Implementing dialog responses in a client application

A dialog node can respond to users with a response that includes text, images, or interactive elements such as clickable options. If you are building your own client application, you must implement the correct display of all response types that are returned by your dialog. For more information about dialog responses, see Responses.

Response output format

By default, responses from a dialog node are specified in the `output.generic` object in the response JSON returned from the `/message` API. The `generic` object contains an array of up to 5 response elements that are intended for any channel. The following JSON example shows a response that includes text and an image:

```
$ {
  "output": {
    "generic": [
      {
        "response_type": "text",
        "text": "OK, here's a picture of a dog."
      },
      {
        "response_type": "image",
        "source": "http://example.com/dog.jpg"
      }
    ],
    "text" : ["OK, here's a picture of a dog."]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Note: As this example shows, the text response (`OK, here's a picture of a dog.`) is also returned in the `output.text` array. The array is included for compatibility with earlier applications that do not support the `output.generic` format.

It is the responsibility of your client application to handle all response types. In this case, your application would need to display the specified text and image to the user.

Response types

Each element of a response is of one of the supported response types. Each response type is specified by using a different set of JSON properties, so the properties that are included for each response vary depending upon the response type. For complete information about the response model of the `/message` API, see the API Reference.

This section describes the available response types and how they are represented in the `/message` API response JSON. If you are using the Watson SDK, you can use the interfaces that are provided for your language to access the same objects.

Note: The examples in this section show the format of the JSON data returned from the `/message API` at run time, and is different from the JSON format that is used to define responses within a dialog node. You can use the format in the examples to update `output.generic` with webhooks. To update `output.generic` with the JSON editor, see Defining responses by using the JSON editor).

Text

The `text` response type is used for ordinary text responses from the dialog:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "text",
        "text": "OK, you want to fly to Boston next Monday."
      }
```

```
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Note: For compatibility, the same text is also included in the `output.text` array in the dialog response.

Image

The `image` response type instructs the client application to display an image, optionally accompanied by a title and description:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "image",
        "source": "http://example.com/image.jpg",
        "title": "Image example",
        "description": "This is an example image"
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Your application is responsible for retrieving the image that is specified by the `source` property and displaying it to the user. If the optional `title` and `description` are provided, your application can display them in whatever way is appropriate (for example, rendering the title after the image and the description as hover text).

Video

The `video` response type instructs the client application to display a video, optionally accompanied by a `title`, `description`, and `alt_text` for accessibility:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "video",
        "source": "http://example.com/video.mp4",
        "title": "Video example",
        "description": "This is an example video",
        "alt_text": "A video showing a great example",
        "channel_options": {
          "chat": {
            "dimensions": {
              "base_height": 180
            }
          }
        }
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Your application is responsible for retrieving the video that is specified by the `source` property and displaying it to the user. If the optional `title` and `description` are provided, your application can display them in whatever way is appropriate.

The optional `channel_options.chat.dimensions.base_height` property takes a number that signifies that the video is rendered at a width of 360 pixels. Your app uses this value to maintain the proper aspect ratio of the video if it is rendered in a nonstandard size.

Audio

The `audio` response type instructs the client application to play an audio file:

```
$ {
  "output": {
```

```
    "generic":[
      {
        "response_type": "audio",
        "source": "http://example.com/audio.mp3",
        "channel_options": {
          "voice_telephony": {
            "loop": true
          }
        }
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Your application is responsible for playing the audio file.

The optional `channel_options.voice_telephony.loop` property takes a Boolean that signifies if the audio file is played as a continuous loop. This option is typically used for hold music that might need to continue for an undefined time.

iframe

The `iframe` response type instructs the client application to display content in an embedded `iframe` element, optionally accompanied by a title:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "iframe",
        "source": "http://example.com/iframe.html",
        "title": "My IFrame"
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Your application is responsible for displaying the `iframe` content. Content in an embedded `iframe` is useful for displaying third-party content, or for content from your own site that you do not want to reauthor using the user_defined response type.

Pause

The `pause` response type instructs the application to wait for a specified interval before the next response:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "pause",
        "time": 500,
        "typing": false
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

This pause might be requested by the dialog to allow time for a request to complete, or to mimic the appearance of a human agent who might pause between responses. The pause can be of any duration up to 10 seconds.

A `pause` response is typically sent in combination with other responses. Your application pauses for the interval that is specified by the `time` property (in milliseconds) before the next response in the array. The optional `typing` property requests that the client application shows a `user is typing` indicator, if supported, to simulate a live agent.

Option

The `option` response type instructs the client application to display a user interface control that enables the user to select from a list of options, and then send input back to the assistant based on the selected option:

```
$ {
  "output": {
    "generic":[
      {
        "response_type": "option",
        "title": "Available options",
        "description": "Please select one of the following options:",
        "preference": "button",
        "options": [
          {
            "label": "Option 1",
            "value": {
              "input": {
                "text": "option 1"
              }
            }
          },
          {
            "label": "Option 2",
            "value": {
              "input": {
                "text": "option 2"
              }
            }
          }
        ]
      }
    ]
  },
  "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```


Your app can display the specified options using any suitable user-interface control (for example, a set of buttons or a drop-down list). The optional `preference` property indicates the preferred type of control your app uses (`button` or `dropdown`), if supported. For the best user experience, a good practice is to present three or fewer options as buttons, and more than three options as a drop-down list.

For each option, the `label` property specifies the label text that appears for the option in the UI control. The `value` property specifies the input that is sent back to the assistant (using the `/message` API) when the user selects the corresponding option.

For an example of implementing `option` responses in a simple client application, see Example: Implementing option responses.

Suggestion

Plus

The `suggestion` response type is used by the disambiguation feature to suggest possible matches to clarify what the user wants to do. A `suggestion` response includes an array of `suggestions` , each one corresponding to a possible matching dialog node:


```
$ {
  "output": {
    "generic": [
      {
        "response_type": "suggestion",
        "title": "Did you mean:",
        "suggestions": [
          {
            "label": "I'd like to order a drink.",
            "value": {
              "intents": [
                {
                  "intent": "order_drink",
                  "confidence": 0.7330395221710206
                }
              ],
              "entities": [],
              "input": {
                "suggestion_id": "576aba3c-85b9-411a-8032-28af2ba95b13",
                "text": "I want to place an order"
              }
            },
```

```
  "output": {
    "text": [
      "I'll get you a drink."
    ],
    "generic": [
      {
        "response_type": "text",
        "text": "I'll get you a drink."
      }
    ],
    "nodes_visited_details": [
      {
        "dialog_node": "node_1_1547675028546",
        "title": "order drink",
        "user_label": "I'd like to order a drink.",
        "conditions": "#order_drink"
      }
    ]
  },
  "source_dialog_node": "root"
},
{
  "label": "I need a drink refill.",
  "value": {
    "intents": [
      {
        "intent": "refill_drink",
        "confidence": 0.2529746770858765
      }
    ],
    "entities": [],
    "input": {
      "suggestion_id": "6583b547-53ff-4e7b-97c6-4d062270abcd",
      "text": "I need a drink refill"
    }
  },
  "output": {
    "text": [
      "I'll get you a refill."
    ],
    "generic": [
      {
        "response_type": "text",
        "text": "I'll get you a refill."
      }
    ],
    "nodes_visited_details": [
      {
        "dialog_node": "node_2_1547675097178",
        "title": "refill drink",
        "user_label": "I need a drink refill.",
        "conditions": "#refill_drink"
      }
    ],
    "source_dialog_node": "root"
  }
 ]
 }
 ],
 },
 "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```


The structure of a `suggestion` response is similar to the structure of an `option` response. As with options, each suggestion includes a `label` that can be displayed to the user and a `value` specifying the input that is sent back to the assistant if the user chooses the corresponding suggestion. To implement `suggestion` responses in your application, you can use the same approach that you would use for `option` responses.

For more information about the disambiguation feature, see Disambiguation.

Search

Tip: This feature is available only to users with a paid plan.

The `search` response type is used by a search skill to return the results from an IBM Watson® Discovery search. A `search` response includes an array of `results`, each of which provides information about a match that is returned from the Discovery search query:


```
$ {
  "output": {
    "generic": [
      {
        "response_type": "search",
        "header": "I found the following information that might be helpful.",
        "results": [
          {
            "title": "About",
            "body": "IBM watsonx Assistant is a cognitive bot that you can customize for your business needs, and deploy across multiple channels to bring help to your customers where and when they need it.",
            "url": "https://cloud.ibm.com/docs/watson-assistant?topic=watson-assistant-about",
            "id": "6682eca3c5b3778ccb730b799a8063f3",
            "result_metadata": {
              "confidence": 0.08401551980328191,
              "score": 0.73975396
            },
            "highlight": {
              "Shortdesc": [
                "IBM <em>watsonx</em> <em>Assistant</em> is a cognitive bot that you can customize for your business needs, and deploy across multiple channels to bring help to your customers where and when they need it."
              ],
              "url": [
                "https://cloud.ibm.com/docs/<em>assistant</em>?topic=<em>watson-assistant</em>-about"
              ],
              "body": [
                "IBM <em>watsonx</em> <em>Assistant</em> is a cognitive bot that you can customize for your business needs, and deploy across multiple channels to bring help to your customers where and when they need it."
              ]
            }
          }
        ]
      }
    ]
  },
  "user_id": "58e1b04e-f4bb-469a-9e4c-dffe1d4ebf23"
}
```


For each search result, the `title`, `body`, and `url` properties include content that is returned from the Discovery query. The search integration configuration determines which fields in the Discovery collection are mapped to these fields in the response. Your application can use these fields to display the results to the user (for example, you might use the `body` text to show an abstract or description of the matching document, and the `url` value to create a link that the user can click to open the document).

In addition, the `header` property provides a message to display to the user about the results of the search. When a search is successful, `header` provides introductory text to be displayed before the search results (for example, `I found the following information that might be helpful.`). Different message text indicates that the search did not return any results, or that the connection to the Discovery service failed. You can customize these messages in the search skill configuration.

User-defined

A user-defined response type can contain up to 5000 KB of data to support a type of a response you implemented in your client. For example, you might define a user-defined response type to display a special color-coded card, or to format data in a table or graphic.

The `user_defined` property of the response is an object that can contain any valid JSON data:


```
$ {
  "output": {
    "generic":[
      {
        "response_type": "user_defined",
        "user_defined": {
```

```
        "field_1": "String value",
        "array_1": [
          1,
          2
        ],
        "object_1": {
          "property_1": "Another string value"
        }
      }
    }
  ]
 },
 "user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```

Your application can parse and display the data in any way you choose.

Example: Implementing option responses

To show how a client application might handle option responses, which prompt the user to select from a list of choices, we can extend the client example that is used in Building a client application. The example is a simplified client app that uses standard input and output to handle three intents (sending a greeting, showing the current time, and exiting from the app):

```
Welcome to the example!
>> hello
Good day to you.
>> what time is it?
The current time is 12:40:42 PM.
>> goodbye
OK! See you later.
```

Note: If you want to try the example code, set up the required workspace and obtain the API details that you need. For more information, see Building a client application.

Receiving an option response

The `option` response can be used when you want to present the user with a finite list of choices, rather than interpreting natural language input. The response can be used in any situation where you want to enable the user to quickly select from a set of unambiguous options.

In our simplified client app, we use this capability to select from a list of the actions the assistant supports (greetings, displaying the time, and exiting). In addition to the three intents previously shown (`#hello`, `#time`, and `#goodbye`), the example workspace supports a fourth intent: `#menu`, which is matched when the use asks to see a list of available actions.

When the workspace recognizes the `#menu` intent, the dialog responds with an `option` response:

```
$ {
"output": {
 "generic": [
  {
   "title": "What do you want to do?",
   "options": [
    {
     "label": "Send greeting",
     "value": {
      "input": {
       "text": "hello"
      }
     }
    },
    {
     "label": "Display the local time",
     "value": {
      "input": {
       "text": "time"
      }
     }
    },
    {
```

```
          "label": "Exit",
          "value": {
            "input": {
              "text": "goodbye"
            }
          }
        }
      ],
      "response_type": "option"
    }
  ],
  "intents": [
    {
      "intent": "menu",
      "confidence": 0.6178638458251953
    }
  ],
  "entities": []
},
"user_id": "faf4a112-f09f-4a95-a0be-43c496e6ac9a"
}
```


The `option` response contains multiple options to be presented to the user. Each option includes two objects, `label` and `value`. The `label` is a user-facing string that identifies the option; the `value` specifies the corresponding message input that is sent back to the assistant if the user chooses the option.

Our client app needs to use the data in this response to build the output we show to the user, and to send the appropriate message to the assistant.

Listing available options

The first step in handling an option response is to display the options to the user, using the text specified by the `label` property of each option. You can display the options using any technique that your application supports, typically a drop-down list or a set of clickable buttons. The optional `preference` property of an option response, if specified, indicates which type of display the application uses if possible.

Our simplified example uses standard input and output, so we don't have access to a real UI. Instead, we present the options as a numbered list.

```
// Option example 1: lists options.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
const { IamAuthenticator } = require('ibm-watson/auth');

// Set up Assistant service wrapper.
const service = new AssistantV2({
  version: '2019-02-28',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  })
});

const assistantId = '{assistant_id}'; // replace with assistant ID
let sessionId;

// Create session.
service
  .createSession({
    assistantId,
  })
  .then(res => {
    sessionId = res.result.session_id;
    sendMessage({
      messageType: 'text',
      text: '',  // start conversation with empty message
    });
  })
  .catch(err => {
    console.log(err); // something went wrong
  });

// Send message to assistant.
function sendMessage(messageInput) {
```

```
  service
    .message({
      assistantId,
      sessionId,
      input: messageInput,
    })
    .then(res => {
      processResponse(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the response.
function processResponse(response) {

  let endConversation = false;

  // Check for client actions requested by the assistant.
  if (response.output.actions) {
    if (response.output.actions[0].type === 'client'){
      if (response.output.actions[0].name === 'display_time') {
        // User asked what time it is, so we output the local system time.
        console.log('The current time is ' + new Date().toLocaleTimeString() + '.');
      } else if (response.output.actions[0].name === 'end_conversation') {
        // User said goodbye, so we're done.
        console.log(response.output.generic[0].text);
        endConversation = true;
      }
    }
  }
} else {
  // Display the output from assistant, if any. Supports only a single
  // response.
  if (response.output.generic) {
    if (response.output.generic.length > 0) {
      switch (response.output.generic[0].response_type) {
        case 'text':
          // It's a text response, so we just display it.
          console.log(response.output.generic[0].text);
          break;
        case 'option':
          // It's an option response, so we'll need to show the user
          // a list of choices.
          console.log(response.output.generic[0].title);
          const options = response.output.generic[0].options;
          // List the options by label.
          for (let i = 0; i < options.length; i++) {
            console.log((i+1).toString() + '. ' + options[i].label);
          }
          break;
      }
    }
  }
}

// If we're not done, prompt for the next round of input.
if (!endConversation) {
  const newMessageFromUser = prompt('>> ');
  newMessageInput = {
    messageType: 'text',
    text: newMessageFromUser,
  }
  sendMessage(newMessageInput);
} else {
  // We're done, so we delete the session.
  service
    .deleteSession({
      assistantId,
      sessionId,
    })
```

```
    .then(res => {
      return;
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
  }
}
```

Let's take a closer look at the code that outputs the response from the assistant. Now, instead of assuming a `text` response, the application supports both the `text` and `option` response types:

```
  // Display the output from assistant, if any. Supports only a single
  // response.
  if (response.output.generic) {
    if (response.output.generic.length > 0) {
      switch (response.output.generic[0].response_type) {
        case 'text':
          // It's a text response, so we just display it.
          console.log(response.output.generic[0].text);
          break;
        case 'option':
          // It's an option response, so we'll need to show the user
          // a list of choices.
          console.log(response.output.generic[0].title);
          const options = response.output.generic[0].options;
          // List the options by label.
          for (let i = 0; i < options.length; i++) {
            console.log((i+1).toString() + '. ' + options[i].label);
          }
          break;
      }
    }
  }
```

If `response_type` = `text`, we display the output, as before. But if `response_type` = `option`, we must do a bit more work. First, we display the value of the `title` property, which serves as lead-in text to introduce the list of options; then, we list the options, using the value of the `label` property to identify each one. (A real-world application would show these labels in a drop-down list or as the labels on clickable buttons.)

You can see the result by triggering the `#menu` intent:

```
Welcome to the example!
>> what are the available actions?
What do you want to do?
1. Send greeting
2. Display the local time
3. Exit
>> 2
Sorry, I have no idea what you're talking about.
>>
```

As you can see, the application is now correctly handling the `option` response by listing the available choices. However, we aren't yet translating the user's choice into meaningful input.

Selecting an option

In addition to the `label`, each option in the response also includes a `value` object, which contains the input data that is sent back to the assistant if the user chooses the corresponding option. The `value.input` object is equivalent to the `input` property of the `/message` API, which means that we can send this object back to the assistant as-is.

We set a new `promptOption` flag when the client receives an `option` response. When this flag is true, we know that we want to take the next round of input from `value.input` rather than accepting natural language text input from the user. Again, we don't have a real user interface, so we prompt the user to select a valid option from the list by number.

```
// Option example 2: sends back selected option value.

const prompt = require('prompt-sync')();
const AssistantV2 = require('ibm-watson/assistant/v2');
```

```
const { IamAuthenticator } = require('ibm-watson/auth');

// Set up Assistant service wrapper.
const service = new AssistantV2({
  version: '2019-02-28',
  authenticator: new IamAuthenticator({
    apikey: '{apikey}', // replace with API key
  })
});

const assistantId = '{assistant_id}'; // replace with assistant ID
let sessionId;

// Create session.
service
  .createSession({
    assistantId,
  })
  .then(res => {
    sessionId = res.result.session_id;
    sendMessage({
      messageType: 'text',
      text: '',  // start conversation with empty message
    });
  })
  .catch(err => {
    console.log(err); // something went wrong
  });

// Send message to assistant.
function sendMessage(messageInput) {
  service
    .message({
      assistantId,
      sessionId,
      input: messageInput,
    })
    .then(res => {
      processResponse(res.result);
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
}

// Process the response.
function processResponse(response) {

  let endConversation = false;
  let promptOption = false;

  // Check for client actions requested by the assistant.
  if (response.output.actions) {
    if (response.output.actions[0].type === 'client'){
      if (response.output.actions[0].name === 'display_time') {
        // User asked what time it is, so we output the local system time.
        console.log('The current time is ' + new Date().toLocaleTimeString() + '.');
      } else if (response.output.actions[0].name === 'end_conversation') {
        // User said goodbye, so we're done.
        console.log(response.output.generic[0].text);
        endConversation = true;
      }
    }
  } else {
    // Display the output from assistant, if any. Supports only a single
    // response.
    if (response.output.generic) {
      if (response.output.generic.length > 0) {
        switch (response.output.generic[0].response_type) {
          case 'text':
            // It's a text response, so we just display it.
```

```
          console.log(response.output.generic[0].text);
          break;
        case 'option':
          // It's an option response, so we'll need to show the user
          // a list of choices.
          console.log(response.output.generic[0].title);
          const options = response.output.generic[0].options;
          // List the options by label.
          for (let i = 0; i < options.length; i++) {
            console.log((i+1).toString() + '. ' + options[i].label);
          }
          promptOption = true;
          break;
      }
    }
  }
}

// If we're not done, prompt for the next round of input.
if (!endConversation) {
  let messageInput;
  if (promptOption == true) {
    // Prompt for a valid selection from the list of options.
    let choice;
    do {
      choice = prompt('? ');
      if (isNaN(choice)) {
        choice = 0;
      }
    } while (choice < 1 || choice > response.output.generic[0].options.length);
    const value = response.output.generic[0].options[choice-1].value;
    // Use message input from the selected option.
    messageInput = value.input;
  } else {
    // We're not showing options, so we just prompt for the next
    // round of input.
    const newText = prompt('>> ');
    messageInput = {
      text: newText,
    }
  }
  sendMessage(messageInput);
} else {
  // We're done, so we delete the session.
  service
    .deleteSession({
      assistantId,
      sessionId,
    })
    .then(res => {
      return;
    })
    .catch(err => {
      console.log(err); // something went wrong
    });
  }
}
```

All that we must do is use the `value.input` object from the selected response as the next round of message input, rather than building a new `input` object using text input. The assistant then responds exactly as if the user typed the input text directly.

```
Welcome to the example!
>> hi
Good day to you.
>> what are the choices?
What do you want to do?
1. Send greeting
2. Display the local time
3. Exit
? 2
```

```
The current time is 1:29:14 PM.
>> bye
OK! See you later.
```

We can now access all of the functions of the assistant either by making natural-language requests or by selecting from a menu of options.

The same approach is used for `suggestion` responses as well. If your plan supports the disambiguation feature, you can use similar logic to prompt users to select from a list when it isn't clear which of several possible options is correct. For more information about the disambiguation feature, see Disambiguation.

Processing input attachments

If you are building a custom channel application by using the REST API, you can add support for sending media files as input attachments.

The request body of the `message` method supports an `attachments` array, which can specify up to 5 media objects. Media objects that are sent in the `attachments` array can be intercepted and processed by a configured premessage webhook.

For detailed information about how to access attachments by using the API, see the API reference.

Examples

The following example shows the request body that is sent to a premessage webhook with a message body that includes the text `Hello` and a JPEG image file as an attachment. The application that receives the webhook request can parse the `attachments` array and process the attachment, optionally modifying the message, before it is processed by the assistant.

```
{
  "event":{
    "name":"message_received"
  },
  "options":{
  },
  "payload":{
    "input":{
      "message_type":"text",
      "text":"Hello",
      "source":{
        "type":"user",
        "id":"00000000000000000000000000000000"
      },
      "options":{
        "suggestion_only":false,
        "return_context":true
      },
      "attachments": [
        {
          "media_type": "image/jpeg",
          "url": "https://example.com/yourphoto.jpeg"
        }
      ]
    },
    "context":{
      "global":{
        "system":{
          "user_id":"00000000000000000000000000000000"
        }
      },
      "integrations":{
        "text_messaging":{
          "assistant_phone_number":"+12223334444",
          "private":{
            "user_phone_number":"+14443332222",
            "request_id":"00000000-0000-0000-0000-000000000000",
            "ip_address":"172.10.10.10"
          }
        }
      }
    }
  }
}
```

This example shows a stateful `/message` request that is sent by a custom channel application and including a single media attachment.

Node

```
service
  .message({
    assistant_id: '{assistant_id}',
    session_id: '{session_id}',
    input: {
      message_type: 'text',
      text: 'Hello',
      attachments: [
        {
          'media_type': 'image/jpeg',
          'url': 'https://example.com/yourphoto.jpeg'
        }
      ]
    }
  })
  .then(res => {
    console.log(JSON.stringify(res, null, 2));
  })
  .catch(err => {
    console.log(err);
  });
```

Python

```
response=service.message(
    assistant_id='{assistant_id}',
    session_id='{session_id}',
    input={
        'message_type': 'text',
        'text': 'Hello',
        'attachments': [
            {
                'media_type': 'image/jpeg',
                'url': 'https://example.com/yourphoto.jpeg'
            }
        ]
    }
).get_result()

print(json.dumps(response, indent=2))
```

Watson SDKs

SDKs abstract much of the complexity associated with application development. By providing programming interfaces in languages that you already know, they can help you get up and running quickly with IBM Watson services.

Supported SDKs

The following Watson SDKs are supported by IBM:

- Java SDK
- Node.js SDK
- Python SDK

Tip: The API reference for each service includes information and examples for these SDKs.

Community SDKs

The following SDKs are available from the Watson community of developers:

- ABAP SDK for IBM Watson, using SAP NetWeaver
- Android SDK

- Go SDK
- Ruby SDK
- Salesforce SDK
- Swift SDK
- Unity SDK

SDK updates and deprecation

The supported Watson SDKs are updated according to the following guidelines.

Semantic versioning

Supported Watson SDKs adhere to semantic versioning with releases labeled as `{major}.{minor}.{patch}`.

Release frequency

SDKs are released independently and might not update on the same schedule.

- The current releases of the Watson SDKs are updated on a 2- to 6-week schedule. These releases are either minor updates or patches that do not include breaking changes. You can update to any version of the SDK with the same major version number.
- Major updates that might include breaking changes are released approximately every 6 months.

Deprecated release

When a major version is released, support continues on the previous major release for 12 months in a deprecation period. The deprecated release might be updated with bug fixes, but no new features will be added and documentation might not be available.

Obsolete release

After the 12-month deprecation period, a release is obsolete. The release might be functional but is unsupported and not updated. Update to the current release.

Extending your assistant with webhooks

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Pre-message webhooks

Making a call before processing a message

A pre-message webhook makes a call to an external service or application every time a customer submits input. The external service can process the message before your assistant processes it.

Add a pre-message webhook to your assistant if you want the webhook to be triggered before each incoming message is processed by your assistant.

> ⚠ **Important:** If you are using a custom channel, the pre-message webhook works with the version 2 of the `/message` API only, stateless and stateful. For more information, see the API reference. All built-in channel integrations use this API.

You can use pre-message webhooks for the following use cases:

- Translate the customer's input to the language that your assistant uses.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

You can use this webhook in coordination with the post-message webhook. For example, the post-message webhook can do things like translate the response back into the customer's language or add back information that was removed for privacy reasons. For more information, see Making a call after processing a message.

> **Note:** For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

Use a dialog webhook if you need to perform a one-time action when needed during a conversation. For example, conditions are met when the assistant collects all required details, such as the account number, user ID, and account secret. For more information, see Making a programmatic call from dialog.

Defining the webhook

You can define one webhook URL to use for preprocessing every incoming message.

Before you begin

The programmatic call to the external service must meet the following requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.
- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).
- The call must return in 30 seconds or less.

Note: If your external service supports GET requests only, or if you need to specify URL parameters dynamically at run time, consider creating an intermediate service that accepts a POST request with a JSON payload that contains any runtime values. The intermediate service can then make a request to the target service, passing these values as URL parameters, and then return the response to the dialog.

Choosing your deployment method

Select the deployment method that you use to view the correct steps for setting a pre-message webhook.

To identify which deployment type you are using, click the **Manage** menu . If you see **Switch to classic experience**, you are using the **new experience**. If you see **Switch to new experience**, you are using the **classic experience**.

Use the following links to know about the procedures based on your deployment type:

- IBM Cloud Pak for Data
- IBM Cloud Pak for Data - classic experience
- watsonx Assistant
- watsonx Assistant - classic experience

Calling a service before processing a message for watsonx Assistant

Use a pre-message webhook to call an external service before your assistant processes a customer's message.

You can use pre-message webhooks for the following use cases:

- Translate the customer's input to the language that your assistant uses.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

Important: This webhook only works with the version 2 of the `/message` API, used by all built-in channels. Custom channels must also use this API.

Learn more

For more information about related features and details, see the following resources.

- Making a call after processing a message
- Calling a service after processing a message for watsonx Assistant
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.

- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).
- The call must return a response in 30 seconds or less.
- If your service supports only GET or needs URL parameters, use a middleware service to handle the POST and forward the data.

Procedure

This section covers the procedure to define, test, and remove pre-message webhooks for watsonx Assistant.

Webhook configuration

To add the webhook details, complete the following steps:

1. From the navigation panel, click **Environments** and open the environment where you want to configure the webhook.
2. Click the icon to open the environment settings.
3. Set the **Pre-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information see, Configuring webhook error handling for preprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, you might write a Cloud Functions web action that checks if a message is in a language other than English, and send it to the Language Translator service to convert it to English. Specify the URL for your web action, as in this example:

   ```
   https://us-south.functions.cloud.ibm.com/api/v1/web/my_org_dev/default/translateToEnglish.json
   ```

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. To configure the authentication for pre-message webhooks, click **Edit authentication**. For detailed instructions, see Defining the authentication method for pre-message and post-message webhooks.
7. In the **Timeout** field, specify the time duration, in seconds, that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Configuring webhook error handling for preprocessing

You can decide whether an error returns in the preprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores errors and processes the message without the webhook result. If preprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If preprocessing is critical before the assistant processes a message, select this option.

> ⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the preprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the message processing.

Testing the webhook

> ⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered when a message is sent to your assistant to be processed.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Error validating webhook response	The webhook's HTTP response body was not a valid `/message` body.
422 Webhook responded with `[500]` status code	There's a problem with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request body of the pre-message webhook so that your external code can process it.

The payload contains the request body of the `/message`, stateful or stateless, version 2 of the API request. The event name `message_received` indicates that the request is generated by the pre-message webhook. For more information about the message request body, see the API reference.

```
{
  "payload" : { Copy of request body sent to /message }
  "event": {
     "name": "message_received"
   }
}
```

Skipping the assistant processing

Enhancements to pre-message webhooks allow watsonx Assistant to skip message processing and directly return the response from the webhook. This functionality is activated by setting the `x-watson-assistant-webhook-return` header in the webhook's HTTP response.

Before you begin

Complete the following steps:

- Include the `x-watson-assistant-webhook-return` header with any value in the HTTP response from your webhook.
- Ensure that the webhook response contains a valid message response, which is formatted according to the watsonx Assistant requirements.

This feature enables the webhook to dynamically control the conversation flow, enabling immediate responses when needed.

Response body

The service that receives the POST request from the webhook must return a JSON object (`Accept: application/json`).

The response body must have the following structure:

```
{
  "payload": {
    ...
  }
}
```

The response `payload` must include the `payload` from the request body. Your code can modify property values or modify context variables, but the returned message payload must follow the `message` method schema. For more information, see API reference.

Example 1

This example shows you how to check the language of the input text, and append the language information to the input text string.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/check_language`
- **Header name**: Content-Type
- **Header value**: application/json

The pre-message webhook calls an IBM Cloud Functions web action name `check_language` .

The node.js code in the `check_language` web action looks as follows.

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
  // Send a request to the Watson Language Translator service to check the language of the input text.
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
        'username': 'apikey',
        'password': 'nnn'
      },
headers: {
   "Content-Type":"text/plain"
},
  body: [
        params.payload.input.text
  ],
  json: true,
};
    return rp(options)
   .then(res => {
      params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
      console.log(JSON.stringify(params))
      //Append "in" plus "the language code" to the input text, surrounded by parentheses.
      const response = {
         body : {
            payload : {
               input : {
                  text : params.payload.input.text + ' ' + '(in ' + res.languages[0].language + ')'
               },
```

```
            },
          },
        };
        return response;
})
}
return {
    body : params
}
};
```

To test the webhook, click **Preview**. Submit the text `Buenos días`. The assistant probably can't understand the input, and returns the response from your *Anything else* node. However, if you go to the Analyze page of your assistant and open **Conversations**, you can see what was submitted. Check the most recent user conversation. The log shows that the user input is `Buenos días (in es)`. The `es` in parentheses represents the language code for Spanish, so the webhook worked and recognized that the submitted text was a Spanish phrase.

Example 2

This example shows you how to check the language of the incoming message, and if it's not English, translate it into English before you submit it to the assistant.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks the language of the incoming text. The second action in the sequence translates the text from its original language into English.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/translation_sequence`
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
headers: {
    "Content-Type":"text/plain"
},
  body: [
          params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
})
}
else {
    params.payload.context.skills["actions skill"].user_defined["language"] = 'none'
    return params
}
};
```

The second web action in the sequence sends the text to the Watson Language Translator service to translate the input text from the language that was identified in the previous web action into English. The translated string is then sent to your assistant instead of the original text.

The node.js code for the second action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
//If the the incoming message is not null and is not English, translate it.
if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
      },
  headers: {
    "Content-Type":"application/json"
  },
      //The body includes the parameters that are required by the Language Translator service, the text to translate and the target language to translate it into.
  body: {
     text: [
        params.payload.input.text
        ],
        target: 'en'
  },
  json: true
};
     return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["original_input"] = params.payload.input.text;
       const response = {
          body : {
             payload : {
                "context" : params.payload.context,
                "input" : {
                   "text" : res.translations[0].translation,
                   "options" : {
                      "export" : true
                      }
                },
             },
          },
       };
    return response
})
}
return {
    body : params
    }
};
```

When you test the webhook in the preview panel, you can submit `Buenos días`, and the assistant responds as if you said `Good morning` in English. In fact, when you check the Analyze page of your assistant and open **Conversations**, the log shows that the user input was `Good morning`.

You can add a post-message webhook to translate the message's response back into the customer's language before it is displayed. For more information, see Example 2.

Example 3

This example shows how to compose a webhook response to let watsonx Assistant skip processing the message and directly return the webhook's response.

Webhook configuration

In the pre-message webhook configuration page, specify the following values:

- **URL**: https://your-webhook-url/webhook_skip
- **Header Name**: Content-Type
- **Header Value**: application/json

The node.js code in the webhook_skip web action looks as follows.

```
function main(params) {
  // Your custom logic to determine the response
  let responseText = "This response is directly from the pre-message webhook.";

  const response = {
    headers: {
      "X-Watson-Assistant-Webhook-Return": "true"
    },
    body: {
      output: {
        generic: [
          {
            response_type: "text",
            text: responseText
          }
        ]
      }
    }
  };

  return response;
}
```

Removing the webhook

If you do not want to preprocess customer input with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to remove the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Pre-message webhook**.
4. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Pre-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service before processing a message for watsonx Assistant - classic experience

Use a pre-message webhook to call an external service before your assistant processes a customer's message.

You can use pre-message webhooks for the following use cases:

- Translate the customer's input to the language that your assistant uses.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

⚠ **Important:** This webhook only works with the version 2 of the `/message` API, used by all built-in channels. Custom channels must also use this API.

Learn more

For more information about related features and details, see the following resources.

- Making a call after processing a message
- Calling a service after processing a message for watsonx Assistant
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.
- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).

- The call must return a response in 30 seconds or less.
- If your service supports only GET or needs URL parameters, use a middleware service to handle the POST and forward the data.

Procedure

This section covers the procedure to define, test, and remove pre-message webhooks for watsonx Assistant - classic experience.

Webhook configuration

To add the webhook details, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Pre-message webhook**.
3. Set the **Pre-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information, see Configuring webhook error handling for preprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, you might write a Cloud Functions web action that checks if a message is in a language other than English, and send it to the Language Translator service to convert it to English. Specify the URL for your web action, as in this example:

   ```
   https://us-south.functions.cloud.ibm.com/api/v1/web/my_org_dev/default/translateToEnglish.json
   ```

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. Fill in the **Secret field**. For more information, see Adding a secret.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.

10. Your webhook details are saved automatically.

Adding a secret

Add a client secret in the **Secret** field to pass with the request for authentication with the external service:

- Enter the key as a text string, such as `purple unicorn`.
- Use a maximum of 1,024 characters.
- Do not use context variables.

The external service is responsible for checking and verifying the secret. If no token is required, specify any string. This field cannot be left empty.

> **Note:** To view the secret as you enter it, click the **Show password** icon before typing. After saving the secret, asterisks replace the string, and you can't view it again.

For more information about how this field is used, see Webhook security.

Webhook security

Authenticate the webhook request by verifying the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call:

- For new webhooks or webhooks updated through **Edit authentication**, the authorization header is ignored.
- For existing webhooks with a saved authentication header, the **Edit authentication** option is disabled.
- Updating an existing webhook to use the new authentication configuration changes its behavior.

If you need to test the JWT verification, you can add code to the external service. For example, if you specify `purple unicorn` in the **Secret** field, you can use the following code:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

Configuring webhook error handling for preprocessing

You can decide whether an error returns in the preprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores errors and processes the message without the webhook result. If preprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If preprocessing is critical before the assistant processes a message, select this option.

> ⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the preprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the message processing.

Testing the webhook

> ⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered when a message is sent to your assistant to be processed.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the

full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Error validating webhook response	The webhook's HTTP response body was not a valid `/message` body.
422 Webhook responded with `[500]` status code	There's a problem with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request body of the pre-message webhook so that your external code can process it.

The payload contains the request body of the `/message`, stateful or stateless, version 2 of the API request. The event name `message_received` indicates that the request is generated by the pre-message webhook. For more information about the message request body, see the API reference.

```
{
  "payload" : { Copy of request body sent to /message }
  "event": {
     "name": "message_received"
   }
}
```

Skipping the assistant processing

Enhancements to pre-message webhooks allow watsonx Assistant - classic experience to skip message processing and directly return the response from the webhook. This functionality is activated by setting the `x-watson-assistant-webhook-return` header in the webhook's HTTP response.

Before you begin

Complete the following steps:

- Include the `x-watson-assistant-webhook-return` header with any value in the HTTP response from your webhook.
- Ensure that the webhook response contains a valid message response, which is formatted according to the watsonx Assistant - classic experience requirements.

This feature enables the webhook to dynamically control the conversation flow, enabling immediate responses when needed.

Response body

The service that receives the POST request from the webhook must return a JSON object (`Accept: application/json`).

The response body must have the following structure:

```
{
  "payload": {
    ...
  }
}
```

The response `payload` must include the `payload` from the request body. Your code can modify property values or modify context variables, but the returned message payload must follow the `message` method schema. For more information, see API reference.

Example 1

This example shows you how to check the language of the input text, and append the language information to the input text string.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/check_language`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The pre-message webhook calls an IBM Cloud Functions web action name `check_language`.

The node.js code in the `check_language` web action looks as follows.

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
  // Send a request to the Watson Language Translator service to check the language of the input text.
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
      },
headers: {
    "Content-Type":"text/plain"
},
  body: [
        params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
       console.log(JSON.stringify(params))
       //Append "in" plus "the language code" to the input text, surrounded by parentheses.
       const response = {
          body : {
             payload : {
                input : {
                   text : params.payload.input.text + ' ' + '(in ' + res.languages[0].language + ')'
                },
             },
          },
       };
       return response;
})
}
return {
   body : params
}
};
```

To test the webhook, click **Preview**. Submit the text `Buenos días`. The assistant probably can't understand the input, and returns the response from your *Anything else* node. However, if you go to the Analyze page of your assistant and open **Conversations**, you can see what was submitted. Check the most recent user conversation. The log shows that the user input is `Buenos días (in es)`. The `es` in parentheses represents the language code for Spanish, so the webhook worked and recognized that the submitted text was a Spanish phrase.

Example 2

This example shows you how to check the language of the incoming message, and if it's not English, translate it into English before you submit it to the assistant.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks the language of the incoming text. The second action in the sequence translates the text from its original language into English.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/translation_sequence`
- **Secret**: None

- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
})
}
else {
    params.payload.context.skills["actions skill"].user_defined["language"] = 'none'
    return params
}
};
```

The second web action in the sequence sends the text to the Watson Language Translator service to translate the input text from the language that was identified in the previous web action into English. The translated string is then sent to your assistant instead of the original text.

The node.js code for the second action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
//If the the incoming message is not null and is not English, translate it.
if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
  headers: {
    "Content-Type":"application/json"
  },
      //The body includes the parameters that are required by the Language Translator service, the text to translate and the target language to translate it into.
  body: {
      text: [
         params.payload.input.text
         ],
         target: 'en'
  },
  json: true
};
     return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["original_input"] = params.payload.input.text;
```

```
        const response = {
            body : {
                payload : {
                    "context" : params.payload.context,
                    "input" : {
                        "text" : res.translations[0].translation,
                        "options" : {
                            "export" : true
                            }
                    },
                },
            },
        };
    return response
})
}
return {
    body : params
    }
};
```

When you test the webhook in the preview panel, you can submit `Buenos días`, and the assistant responds as if you said `Good morning` in English. In fact, when you check the Analyze page of your assistant and open **Conversations**, the log shows that the user input was `Good morning`.

You can add a post-message webhook to translate the message's response back into the customer's language before it is displayed. For more information, see Example 2.

Example 3

This example shows how to compose a webhook response to let watsonx Assistant - classic experience skip processing the message and directly return the webhook's response.

Webhook configuration

In the pre-message webhook configuration page, specify the following values:

- **URL**: https://your-webhook-url/webhook_skip
- **Secret**: None
- **Header Name**: Content-Type
- **Header Value**: application/json

The node.js code in the webhook_skip web action looks as follows.

```
function main(params) {
  // Your custom logic to determine the response
  let responseText = "This response is directly from the pre-message webhook.";

  const response = {
    headers: {
      "X-Watson-Assistant-Webhook-Return": "true"
    },
    body: {
      output: {
        generic: [
          {
            response_type: "text",
            text: responseText
          }
        ]
      }
    }
  };

  return response;
}
```

Removing the webhook

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Pre-message webhook**.
3. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Pre-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service before processing a message for IBM Cloud Pak for Data

Use a pre-message webhook to call an external service before your assistant processes a customer's message.

You can use pre-message webhooks for the following use cases:

- Translate the customer's input to the language that your assistant uses.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

⚠ **Important:** This webhook only works with the version 2 of the `/message` API, used by all built-in channels. Custom channels must also use this API.

Learn more

For more information about related features and details, see the following resources.

- Making a call after processing a message
- Calling a service after processing a message for IBM Cloud Pak for Data
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.
- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).
- The call must return a response in 30 seconds or less.
- If your service supports only GET or needs URL parameters, use a middleware service to handle the POST and forward the data.

Procedure

This section covers the procedure to define, test, and remove pre-message webhooks for Cloud Pak for Data.

- Webhook configuration
- Configuring webhook error handling for preprocessing
- Testing the webhook
- Troubleshooting the webhook
- Example request body
- Skipping the assistant processing
- Response body
- Example 1
- Example 2
- Example 3
- Removing the webhook

Webhook configuration

To add the webhook details, complete the following steps:

1. From the navigation panel, click **Environments** and open the environment where you want to configure the webhook.

2. Click the ⚙ icon to open the environment settings.
3. Set the **Pre-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information see, Configuring webhook error handling for preprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, you might write a Cloud Functions web action that checks if a message is in a language other than English, and send it to the Language Translator service to convert it to English. Specify the URL for your web action, as in this example:

   ```
   https://us-south.functions.cloud.ibm.com/api/v1/web/my_org_dev/default/translateToEnglish.json
   ```

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. To configure the authentication for pre-message webhooks, click **Edit authentication**. For detailed instructions, see Defining the authentication method for pre-message and post-message webhooks.
7. In the **Timeout** field, specify the time duration, in seconds, that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

   Header example
9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Configuring webhook error handling for preprocessing

You can decide whether an error returns in the preprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores errors and processes the message without the webhook result. If preprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If preprocessing is critical before the assistant processes a message, select this option.

⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the preprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the message processing.

Testing the webhook

⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered when a message is sent to your assistant to be processed.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the

full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Error validating webhook response	The webhook's HTTP response body was not a valid `/message` body.
422 Webhook responded with `[500]` status code	There's a problem with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request body of the pre-message webhook so that your external code can process it.

The payload contains the request body of the `/message`, stateful or stateless, version 2 of the API request. The event name `message_received` indicates that the request is generated by the pre-message webhook. For more information about the message request body, see the API reference.

```
{
  "payload" : { Copy of request body sent to /message }
  "event": {
     "name": "message_received"
   }
}
```

Skipping the assistant processing

Enhancements to pre-message webhooks allow Cloud Pak for Data to skip message processing and directly return the response from the webhook. This functionality is activated by setting the `x-watson-assistant-webhook-return` header in the webhook's HTTP response.

Before you begin

Complete the following steps:

- Include the `x-watson-assistant-webhook-return` header with any value in the HTTP response from your webhook.
- Ensure that the webhook response contains a valid message response, which is formatted according to the Cloud Pak for Data requirements.

This feature enables the webhook to dynamically control the conversation flow, enabling immediate responses when needed.

Response body

The service that receives the POST request from the webhook must return a JSON object (`Accept: application/json`).

The response body must have the following structure:

```
{
  "payload": {
    ...
  }
}
```

The response `payload` must include the `payload` from the request body. Your code can modify property values or modify context variables, but the returned message payload must follow the `message` method schema. For more information, see API reference.

Example 1

This example shows you how to check the language of the input text, and append the language information to the input text string.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/check_language`
- **Header name**: Content-Type
- **Header value**: application/json

The pre-message webhook calls an IBM Cloud Functions web action name `check_language`.

The node.js code in the `check_language` web action looks as follows.

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
  // Send a request to the Watson Language Translator service to check the language of the input text.
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
      },
headers: {
   "Content-Type":"text/plain"
},
  body: [
        params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
       console.log(JSON.stringify(params))
       //Append "in" plus "the language code" to the input text, surrounded by parentheses.
       const response = {
          body : {
             payload : {
                input : {
                   text : params.payload.input.text + ' ' + '(in ' + res.languages[0].language + ')'
                },
             },
          },
       };
       return response;
})
}
return {
    body : params
}
};
```

To test the webhook, click **Preview**. Submit the text `Buenos días`. The assistant probably can't understand the input, and returns the response from your *Anything else* node. However, if you go to the Analyze page of your assistant and open **Conversations**, you can see what was submitted. Check the most recent user conversation. The log shows that the user input is `Buenos días (in es)`. The `es` in parentheses represents the language code for Spanish, so the webhook worked and recognized that the submitted text was a Spanish phrase.

Example 2

This example shows you how to check the language of the incoming message, and if it's not English, translate it into English before you submit it to the assistant.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks the language of the incoming text. The second action in the sequence translates the text from its original language into English.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/translation_sequence`
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
       params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
       console.log(JSON.stringify(params))
       return params;
})
}
else {
    params.payload.context.skills["actions skill"].user_defined["language"] = 'none'
    return params
}
};
```

The second web action in the sequence sends the text to the Watson Language Translator service to translate the input text from the language that was identified in the previous web action into English. The translated string is then sent to your assistant instead of the original text.

The node.js code for the second action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
//If the the incoming message is not null and is not English, translate it.
if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
  headers: {
    "Content-Type":"application/json"
  },
      //The body includes the parameters that are required by the Language Translator service, the text to translate and the target language to translate it into.
  body: {
      text: [
         params.payload.input.text
         ],
         target: 'en'
  },
  json: true
};
     return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["original_input"] = params.payload.input.text;
       const response = {
          body : {
             payload : {
```

```
                "context" : params.payload.context,
                "input" : {
                    "text" : res.translations[0].translation,
                    "options" : {
                        "export" : true
                        }
                },
            },
        },
    };
  return response
})
}
return {
  body : params
  }
};
```

When you test the webhook in the preview panel, you can submit `Buenos días`, and the assistant responds as if you said `Good morning` in English. In fact, when you check the Analyze page of your assistant and open **Conversations**, the log shows that the user input was `Good morning`.

You can add a post-message webhook to translate the message's response back into the customer's language before it is displayed. For more information, see Example 2.

Example 3

This example shows how to compose a webhook response to let Cloud Pak for Data skip processing the message and directly return the webhook's response.

Webhook configuration

In the pre-message webhook configuration page, specify the following values:

- **URL**: https://your-webhook-url/webhook_skip
- **Header Name**: Content-Type
- **Header Value**: application/json

The node.js code in the webhook_skip web action looks as follows.

```
function main(params) {
  // Your custom logic to determine the response
  let responseText = "This response is directly from the pre-message webhook.";

  const response = {
    headers: {
      "X-Watson-Assistant-Webhook-Return": "true"
    },
    body: {
      output: {
        generic: [
          {
            response_type: "text",
            text: responseText
          }
        ]
      }
    }
  };

  return response;
}
```

Removing the webhook

If you do not want to preprocess customer input with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to remove the webhook.
2. Click the ⚙ icon to open the environment settings.

3. On the **Environment settings** page, click **Pre-message webhook**.
4. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Pre-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service before processing a message for IBM Cloud Pak for Data - classic experience

Use a pre-message webhook to call an external service before your assistant processes a customer's message.

You can use pre-message webhooks for the following use cases:

- Translate the customer's input to the language that your assistant uses.
- Check for and remove any personally identifiable information, such as an email address or social security number that a customer might submit.

⚠ **Important:** This webhook only works with the version 2 of the `/message` API, used by all built-in channels. Custom channels must also use this API.

Learn more

For more information about related features and details, see the following resources.

- Making a call after processing a message
- Calling a service after processing a message for IBM Cloud Pak for Data - classic experience
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.
- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).
- The call must return a response in 30 seconds or less.
- If your service supports only GET or needs URL parameters, use a middleware service to handle the POST and forward the data.

Procedure

This section covers the procedure to define, test, and remove pre-message webhooks for Cloud Pak for Data - classic experience.

- **Webhook configuration**
- Adding a secret
- Webhook security
- Configuring webhook error handling for preprocessing
- Testing the webhook
- Troubleshooting the webhook
- Example request body
- Skipping the assistant processing
- Response body
- Example 1
- Example 2
- Example 3
- Removing the webhook

Webhook configuration

To add the webhook details, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.

2. Click **Webhooks** > **Pre-message webhook**.
3. Set the **Pre-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information, see Configuring webhook error handling for preprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, you might write a Cloud Functions web action that checks if a message is in a language other than English, and send it to the Language Translator service to convert it to English. Specify the URL for your web action, as in this example:

   ```
   https://us-south.functions.cloud.ibm.com/api/v1/web/my_org_dev/default/translateToEnglish.json
   ```

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. Fill in the **Secret field**. For more information, see Adding a secret.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

   Header example
9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Adding a secret

Add a client secret in the **Secret** field to pass with the request for authentication with the external service:

- Enter the key as a text string, such as `purple unicorn`.
- Use a maximum of 1,024 characters.
- Do not use context variables.

The external service is responsible for checking and verifying the secret. If no token is required, specify any string. This field cannot be left empty.

> **Note:** To view the secret as you enter it, click the **Show password** icon before typing. After saving the secret, asterisks replace the string, and you can't view it again.

For more information about how this field is used, see Webhook security.

Webhook security

Authenticate the webhook request by verifying the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call:

- For new webhooks or webhooks updated through **Edit authentication**, the authorization header is ignored.
- For existing webhooks with a saved authentication header, the **Edit authentication** option is disabled.
- Updating an existing webhook to use the new authentication configuration changes its behavior.

If you need to test the JWT verification, you can add code to the external service. For example, if you specify `purple unicorn` in the **Secret** field, you can use the following code:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

Configuring webhook error handling for preprocessing

You can decide whether an error returns in the preprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores errors and processes the message without the webhook result. If preprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If preprocessing is critical before the assistant processes a message, select this option.

> ⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the preprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the message processing.

Testing the webhook

> ⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered when a message is sent to your assistant to be processed.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else` . If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Error validating webhook response	The webhook's HTTP response body was not a valid `/message` body.
422 Webhook responded with `[500]` status code	There's a problem with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request body of the pre-message webhook so that your external code can process it.

The payload contains the request body of the `/message` , stateful or stateless, version 2 of the API request. The event name `message_received` indicates that the request is generated by the pre-message webhook. For more information about the message request body, see the API reference.

```
{
  "payload" : { Copy of request body sent to /message }
  "event": {
     "name": "message_received"
```

```
  }
}
```

Skipping the assistant processing

Enhancements to pre-message webhooks allow Cloud Pak for Data - classic exerience to skip message processing and directly return the response from the webhook. This functionality is activated by setting the `x-watson-assistant-webhook-return` header in the webhook's HTTP response.

Before you begin

Complete the following steps:

- Include the `x-watson-assistant-webhook-return` header with any value in the HTTP response from your webhook.
- Ensure that the webhook response contains a valid message response, which is formatted according to the Cloud Pak for Data - classic exerience requirements.

This feature enables the webhook to dynamically control the conversation flow, enabling immediate responses when needed.

Response body

The service that receives the POST request from the webhook must return a JSON object (`Accept: application/json`).

The response body must have the following structure:

```
{
  "payload": {
    ...
  }
}
```

The response `payload` must include the `payload` from the request body. Your code can modify property values or modify context variables, but the returned message payload must follow the `message` method schema. For more information, see API reference.

Example 1

This example shows you how to check the language of the input text, and append the language information to the input text string.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/check_language`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The pre-message webhook calls an IBM Cloud Functions web action name `check_language` .

The node.js code in the `check_language` web action looks as follows.

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
  // Send a request to the Watson Language Translator service to check the language of the input text.
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
      },
headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.input.text
  ],
  json: true,
};
```

```
    return rp(options)
   .then(res => {
      params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
      console.log(JSON.stringify(params))
      //Append "in" plus "the language code" to the input text, surrounded by parentheses.
      const response = {
         body : {
            payload : {
               input : {
                  text : params.payload.input.text + ' ' + '(in ' + res.languages[0].language + ')'
               },
            },
         },
      };
      return response;
})
}
return {
   body : params
}
};
```

To test the webhook, click **Preview**. Submit the text `Buenos días`. The assistant probably can't understand the input, and returns the response from your *Anything else* node. However, if you go to the Analyze page of your assistant and open **Conversations**, you can see what was submitted. Check the most recent user conversation. The log shows that the user input is `Buenos días (in es)`. The `es` in parentheses represents the language code for Spanish, so the webhook worked and recognized that the submitted text was a Spanish phrase.

Example 2

This example shows you how to check the language of the incoming message, and if it's not English, translate it into English before you submit it to the assistant.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks the language of the incoming text. The second action in the sequence translates the text from its original language into English.

In the pre-message webhook configuration page, the following values are specified:

- **URL**: `https://us-south.functions.appdomain.cloud/api/v1/web/e97d2516-5ce4-4fd9-9d05-acc3dd8ennn/default/translation_sequence`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
if (params.payload.input.text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
       },
headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.input.text
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills["actions skill"].user_defined["language"] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
```

```
})
}
else {
    params.payload.context.skills["actions skill"].user_defined["language"] = 'none'
    return params
}
};
```

The second web action in the sequence sends the text to the Watson Language Translator service to translate the input text from the language that was identified in the previous web action into English. The translated string is then sent to your assistant instead of the original text.

The node.js code for the second action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
//If the the incoming message is not null and is not English, translate it.
if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
        'username': 'apikey',
        'password': 'nnn'
      },
  headers: {
    "Content-Type":"application/json"
  },
     //The body includes the parameters that are required by the Language Translator service, the text to translate and the target language to translate it into.
  body: {
     text: [
        params.payload.input.text
        ],
        target: 'en'
  },
  json: true
};
    return rp(options)
  .then(res => {
      params.payload.context.skills["actions skill"].user_defined["original_input"] = params.payload.input.text;
      const response = {
          body : {
              payload : {
                  "context" : params.payload.context,
                  "input" : {
                      "text" : res.translations[0].translation,
                      "options" : {
                          "export" : true
                      }
                  },
              },
          },
      };
   return response
})
}
return {
   body : params
   }
};
```

When you test the webhook in the preview panel, you can submit `Buenos días`, and the assistant responds as if you said `Good morning` in English. In fact, when you check the Analyze page of your assistant and open **Conversations**, the log shows that the user input was `Good morning`.

You can add a post-message webhook to translate the message's response back into the customer's language before it is displayed. For more information, see Example 2.

Example 3

This example shows how to compose a webhook response to let Cloud Pak for Data - classic experience skip processing the message and directly return

the webhook's response.

Webhook configuration

In the pre-message webhook configuration page, specify the following values:

- **URL**: https://your-webhook-url/webhook_skip
- **Secret**: None
- **Header Name**: Content-Type
- **Header Value**: application/json

The node.js code in the webhook_skip web action looks as follows.

```
function main(params) {
  // Your custom logic to determine the response
  let responseText = "This response is directly from the pre-message webhook.";

  const response = {
    headers: {
      "X-Watson-Assistant-Webhook-Return": "true"
    },
    body: {
      output: {
        generic: [
          {
            response_type: "text",
            text: responseText
          }
        ]
      }
    }
  };

  return response;
}
```

Removing the webhook

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Pre-message webhook**.
3. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Pre-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Post-message webhooks

Making a call after processing a message

A post-message webhook calls an external service or application every time that the assistant renders a response. The external service can process the assistant's output before it is sent to the channel.

You can add a post-message webhook to your assistant if you want to trigger the webhook before each message response is shown to the customer.

You can use a post-message webhook to do things like extract custom responses from an external content repository. For example, you can define actions with custom IDs in the responses instead of text. The post-message webhook can pass these IDs to an external database to retrieve stored text responses.

You can use this webhook in coordination with the pre-message webhook. For example, if you use the pre-message webhook to strip personally identifiable information from the customer's input, you can use the post-message webhook to add it back. If you use the pre-message webhook to translate the customer's input to the assistant's language, you can use the post-message webhook to translate the response into the customer's language before returning it. For more information, see Making a call before processing a message .

Note: For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

Defining the webhook

You can define one webhook URL to use for processing every message response before it is sent to the channel and shown to the customer.

Before you begin

The programmatic call to the external service must meet these requirements:

- Do not set up and test your webhook in a production environment where the assistant is deployed and is interacting with customers.
- The call must be a POST HTTP request.
- The call must be completed in 30 seconds or less.
- The format of the request and response must be in JSON. For example, `Content-Type: application/json`.

Note: Use a dialog webhook if you need to perform a one-time action when needed during a conversation. For example, conditions are met when the assistant collects all required details, such as the account number, user ID, and account secret. For more information, see Making a programmatic call from dialog.

Choosing your deployment method

Select the deployment method that you use to view the correct steps for setting a post-message webhook.

To identify which deployment type you are using, click the **Manage** menu. If you see **Switch to classic experience**, you are using the **new experience**. If you see **Switch to new experience**, you are using the **classic experience**.

Use the following links to know about the procedures based on your deployment type:

- IBM Cloud Pak for Data
- IBM Cloud Pak for Data - classic experience
- watsonx Assistant
- watsonx Assistant - classic experience

Calling a service after processing a message for watsonx Assistant

Use a post-message webhook to call an external service after your assistant generates a response.

You can use post-message webhooks for the following use cases:

- Retrieve a response from an external source by using custom action IDs.
- Translate the assistant's response to the user's language.
- Reinsert personal data that was removed earlier for privacy.

Learn more

For more information about related features and details, see the following resources:

- Making a call before processing a message
- Calling a service before processing a message for watsonx Assistant
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up or test the webhook in a production environment.
- The call must be a POST HTTP request.
- The request and response must use JSON (Content-Type: application/json).
- The response must return within 30 seconds.

Procedure

This section covers the procedure to define, test, and remove post-message webhooks for watsonx Assistant.

Webhook configuration

To add the webhook details, complete the following steps:

1. From the navigation panel, click **Environments** and open the environment where you want to configure the webhook.
2. Click the ⚙ icon to open the environment settings.
3. Set the **Post-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information, see Configuring webhook error handling for postprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, maybe you store your assistant's responses in a separate content management system. When the assistant understands the input, the processed action returns a unique ID that corresponds to a response in your CMS. To call a service that retrieves a response from your CMS for a given unique ID, specify the URL for your service instance. For example, `https://example.com/get_answer`.

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. To configure the authentication for post-message webhooks, click **Edit authentication**. For detailed instructions, see Defining the authentication method for pre-message and post-message webhooks.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Configuring webhook error handling for postprocessing

You can decide whether an error returns in the postprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores the errors and processes the message without the webhook result. If postprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If postprocessing is critical after the assistant sends a response, select this option.

⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the postprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the response processing.

Testing the webhook

⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered only when your assistant processes a message and a response is ready to be returned to the channel.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Webhook responded with `[500]` status code	A problem occurred with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request post-message webhook body so that your external code can process it.

The payload contains the response body that your assistant returns for the version 2 of the `/message`, stateful and stateless, API call. The event name `message_processed` indicates that the post-message webhook generates the request. For more information about the message request body, see the API reference.

The following sample shows how a simple request body is formatted:

```
{
 "event": {
    "name": "message_processed"
},
"options": {},
"payload": {
  "output": {
     "intents": [
        {
           "intent": "General_Greetings",
           "confidence": 1
        }
     ],
     "entities": [],
     "generic": [
        {
           "response_type": "text",
           "text": "Hello. Good evening"
        }
     ]
  },
  "user_id": "test user",
  "context": {
     "global": {
        "system": {
           "user_id": "test user",
           "turn_count": 11
        },
```

```
        "session_id": "sxxx"
      },
      "skills": {
        "actions skill": {
          "user_defined": {
            "var": "anthony"
          },
          "system": {
            "state": "nnn"
          }
        }
      }
    }
}
```

Example 1

This example shows how to add `y'all` to the end of each response from the assistant.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Header name**: Content-Type
- **Header value**: application/json

The post-message webhook calls an IBM Cloud Functions web action name `add_southern_charm`.

The node.js code in the `add_southern_charm` web action looks as follows:

```
function main(params) {
  console.log(JSON.stringify(params))
  if (params.payload.output.generic[0].text !== '') {
      //Get the length of the input text
        var length = params.payload.output.generic[0].text.length;
        //create a substring that removes the last character from the input string, which is typically punctuation.
        var revision = params.payload.output.generic[0].text.substring(0,length-1);
        const response = {
            body : {
                payload : {
                    output : {
                        generic : [
                            {
                                //Replace the input text with your shortened revision and append y'all to it.
                              "response_type": "text",
                              "text": revision + ', ' + 'y\'all.'
                            }
                        ],
                    },
                },
            },
        };
        return response;
  }
  else {
    return {
        body : params
    }
  }
}
```

Example 2

This example shows how to translate a message response back to the customer's language. It works only if you perform the steps in Example 2 to define a pre-message webhook that translates the original message into English.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks for the language of the original incoming text, which you stored in a context variable named `original_input` in the pre-message webhook code. The second action in the sequence translates the dialog response text from English into the original language that was used by the customer.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))

if (params.payload.output.generic[0].text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnnn'
      },
  headers: {
    "Content-Type":"text/plain"
},
  body: [
        params.payload.context.skills['actions skill'].user_defined.original_input
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills['actions skill'].user_defined['language'] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
})
}
else {
    params.payload.context.skills['actions skill'].user_defined['language'] = 'none';
    return param
}
};
```

The second web action in the sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
    if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
  const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnn'
      },
  body: {
      text: [
          params.payload.output.generic[0].text
          ],
          target: params.payload.context.skills["actions skill"].user_defined.language
  },
  json: true
  };
    return rp(options)
    .then(res => {
        params.payload.context.skills["actions skill"].user_defined["original_output"] = params.payload.output.generic[0].text;
        params.payload.output.generic[0].text = res.translations[0].translation;
        return {
          body : params
        }
})
```

```
  }
  return {
    body : params
  }
};
```

Removing the webhook

If you do not want to process message responses with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to remove the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Post-message webhook**.
4. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Post-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service after processing a message for watsonx Assistant - classic experience

Use a post-message webhook to call an external service after your assistant generates a response.

You can use post-message webhooks for the following use cases:

- Retrieve a response from an external source by using custom action IDs.
- Translate the assistant's response to the user's language.
- Reinsert personal data that was removed earlier for privacy.

Learn more

For more information about related features and details, see the following resources:

- Making a call before processing a message
- Calling a service before processing a message for watsonx Assistant - classic experience
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up or test the webhook in a production environment.
- The call must be a POST HTTP request.
- The request and response must use JSON (Content-Type: application/json).
- The response must return within 30 seconds.

Procedure

This section covers the procedure to define, test, and remove post-message webhooks for watsonx Assistant - classic experience.

Webhook configuration

To add the webhook details, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Post-message webhook**.
3. Set the **Post-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information, see Configuring webhook error handling for postprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, maybe you store your assistant's responses in a separate content management system. When the assistant understands the input, the processed action returns a unique ID that corresponds to a response in your CMS. To call a service that retrieves a response from your CMS for a given unique ID, specify the URL for your service instance. For example, `https://example.com/get_answer`.

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. Fill in the **Secret field**. For more information, see Adding a secret.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   Note: The service automatically sends an Authorization header with a JWT. If you want to handle authorization yourself, add your own authorization header and the service uses it instead.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Adding a secret

Add a client secret in the **Secret** field to pass with the request for authentication with the external service:

- Enter the key as a text string, such as `purple unicorn`.
- Use a maximum of 1,024 characters.
- Do not use context variables.

The external service is responsible for checking and verifying the secret. If no token is required, specify any string. This field cannot be left empty.

Note: To view the secret as you enter it, click the **Show password** icon before typing. After saving the secret, asterisks replace the string, and you can't view it again.

For more information about how this field is used, see Webhook security.

Webhook security

Authenticate the webhook request by verifying the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically

generates a JWT and sends it in the `Authorization` header with each webhook call:

- For new webhooks or webhooks updated through **Edit authentication**, the authorization header is ignored.
- For existing webhooks with a saved authentication header, the **Edit authentication** button is disabled.
- Updating an existing webhook to use the new authentication configuration changes its behavior.

If you need to test the JWT verification, you can add code to the external service. For example, if you specify `purple unicorn` in the **Secret** field, you can use the following code:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

Configuring webhook error handling for postprocessing

You can decide whether an error returns in the postprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores the errors and processes the message without the webhook result. If postprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If postprocessing is critical after the assistant sends a response, select this option.

⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the postprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the response processing.

Testing the webhook

⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered only when your assistant processes a message and a response is ready to be returned to the channel.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else` . If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Webhook responded with `[500]` status code	A problem occurred with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request post-message webhook body so that your external code can process it.

The payload contains the response body that your assistant returns for the version 2 of the `/message` , stateful and stateless, API call. The event name `message_processed` indicates that the post-message webhook generates the request. For more information about the message request body, see the API

reference.

The following sample shows how a simple request body is formatted:

```
{
 "event": {
    "name": "message_processed"
},
"options": {},
"payload": {
    "output": {
        "intents": [
            {
                "intent": "General_Greetings",
                "confidence": 1
            }
        ],
        "entities": [],
        "generic": [
            {
                "response_type": "text",
                "text": "Hello. Good evening"
            }
        ]
    },
    "user_id": "test user",
    "context": {
        "global": {
            "system": {
                "user_id": "test user",
                "turn_count": 11
            },
            "session_id": "sxxx"
        },
        "skills": {
            "actions skill": {
                "user_defined": {
                    "var": "anthony"
                },
                "system": {
                    "state": "nnn"
                }
            }
        }
    }
}
```

Example 1

This example shows how to add `y'all` to the end of each response from the assistant.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The post-message webhook calls an IBM Cloud Functions web action name `add_southern_charm`.

The node.js code in the `add_southern_charm` web action looks as follows:

```
function main(params) {
  console.log(JSON.stringify(params))
  if (params.payload.output.generic[0].text !== '') {
      //Get the length of the input text
        var length = params.payload.output.generic[0].text.length;
        //create a substring that removes the last character from the input string, which is typically punctuation.
        var revision = params.payload.output.generic[0].text.substring(0,length-1);
        const response = {
```

```
          body : {
              payload : {
                  output : {
                      generic : [
                          {
                              //Replace the input text with your shortened revision and append y'all to it.
                            "response_type": "text",
                            "text": revision + ', ' + 'y\'all.'
                          }
                      ],
                  },
              },
          },
      };
      return response;
  }
  else {
    return {
        body : params
    }
  }
}
```

Example 2

This example shows how to translate a message response back to the customer's language. It works only if you perform the steps in Example 2 to define a pre-message webhook that translates the original message into English.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks for the language of the original incoming text, which you stored in a context variable named `original_input` in the pre-message webhook code. The second action in the sequence translates the dialog response text from English into the original language that was used by the customer.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))

if (params.payload.output.generic[0].text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnnn'
       },
  headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.context.skills['actions skill'].user_defined.original_input
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills['actions skill'].user_defined['language'] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
})
}
```

```
else {
    params.payload.context.skills['actions skill'].user_defined['language'] = 'none';
    return param
}
};
```

The second web action in the sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
    if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
    const options = { method: 'POST',
    url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
    auth: {
            'username': 'apikey',
            'password': 'nnn'
        },
    body: {
        text: [
            params.payload.output.generic[0].text
            ],
            target: params.payload.context.skills["actions skill"].user_defined.language
    },
    json: true
    };
      return rp(options)
      .then(res => {
          params.payload.context.skills["actions skill"].user_defined["original_output"] = params.payload.output.generic[0].text;
          params.payload.output.generic[0].text = res.translations[0].translation;
          return {
            body : params
          }
  })
  }
  return {
    body : params
  }
};
```

Removing the webhook

If you do not want to process message responses with a webhook, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Post-message webhook**.
3. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Post-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service after processing a message for IBM Cloud Pak for Data

Use a post-message webhook to call an external service after your assistant generates a response.

You can use post-message webhooks for the following use cases:

- Retrieve a response from an external source by using custom action IDs.
- Translate the assistant's response to the user's language.
- Reinsert personal data that was removed earlier for privacy.

Learn more

For more information about related features and details, see the following resources:

- Making a call before processing a message

- Calling a service before processing a message for IBM Cloud Pak for Data
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up or test the webhook in a production environment.
- The call must be a POST HTTP request.
- The request and response must use JSON (Content-Type: application/json).
- The response must return within 30 seconds.

Procedure

This section covers the procedure to define, test, and remove post-message webhooks for Cloud Pak for Data.

- **Webhook configuration**
- **Configuring webhook error handling for postprocessing**
- **Testing the webhook**
- **Troubleshooting the webhook**
- **Example request body**
- **Example 1**
- **Example 2**
- **Removing the webhook**

Webhook configuration

To add the webhook details, complete the following steps:

1. From the navigation panel, click **Environments** and open the environment where you want to configure the webhook.
2. Click the icon to open the environment settings.
3. Set the **Post-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
    - Continue processing user input without webhook update if there is an error.
    - Return an error to the client if the webhook call fails.

    For more information, see Configuring webhook error handling for postprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

    For example, maybe you store your assistant's responses in a separate content management system. When the assistant understands the input, the processed action returns a unique ID that corresponds to a response in your CMS. To call a service that retrieves a response from your CMS for a given unique ID, specify the URL for your service instance. For example, `https://example.com/get_answer`.

    You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. To configure the authentication for post-message webhooks, click **Edit authentication**. For detailed instructions, see Defining the authentication method for pre-message and post-message webhooks.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

    If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned is in JSON format.

Header name	Header value

Content-Type	application/json

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.
10. Your webhook details are saved automatically.

Configuring webhook error handling for postprocessing

You can decide whether an error returns in the postprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores the errors and processes the message without the webhook result. If postprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If postprocessing is critical after the assistant sends a response, select this option.

⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the postprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the response processing.

Testing the webhook

⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered only when your assistant processes a message and a response is ready to be returned to the channel.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Webhook responded with `[500]` status code	A problem occurred with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request post-message webhook body so that your external code can process it.

The payload contains the response body that your assistant returns for the version 2 of the `/message`, stateful and stateless, API call. The event name `message_processed` indicates that the post-message webhook generates the request. For more information about the message request body, see the API reference.

The following sample shows how a simple request body is formatted:

```
{
 "event": {
    "name": "message_processed"
},
"options": {},
"payload": {
    "output": {
```

```
    "intents": [
      {
        "intent": "General_Greetings",
        "confidence": 1
      }
    ],
    "entities": [],
    "generic": [
      {
        "response_type": "text",
        "text": "Hello. Good evening"
      }
    ]
},
"user_id": "test user",
"context": {
    "global": {
      "system": {
        "user_id": "test user",
        "turn_count": 11
      },
      "session_id": "sxxx"
    },
    "skills": {
      "actions skill": {
        "user_defined": {
          "var": "anthony"
        },
        "system": {
          "state": "nnn"
        }
      }
    }
  }
}
```


Example 1

This example shows how to add `y'all` to the end of each response from the assistant.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Header name**: Content-Type
- **Header value**: application/json

The post-message webhook calls an IBM Cloud Functions web action name `add_southern_charm`.

The node.js code in the `add_southern_charm` web action looks as follows:

```
function main(params) {
  console.log(JSON.stringify(params))
  if (params.payload.output.generic[0].text !== '') {
      //Get the length of the input text
      var length = params.payload.output.generic[0].text.length;
      //create a substring that removes the last character from the input string, which is typically punctuation.
      var revision = params.payload.output.generic[0].text.substring(0,length-1);
      const response = {
          body : {
              payload : {
                  output : {
                      generic : [
                          {
                              //Replace the input text with your shortened revision and append y'all to it.
                            "response_type": "text",
                            "text": revision + ', ' + 'y\'all.'
                          }
                      ],
                  },
              },
```

```
        },
      };
      return response;
  }
  else {
    return {
      body : params
    }
  }
}
```

Example 2

This example shows how to translate a message response back to the customer's language. It works only if you perform the steps in Example 2 to define a pre-message webhook that translates the original message into English.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks for the language of the original incoming text, which you stored in a context variable named `original_input` in the pre-message webhook code. The second action in the sequence translates the dialog response text from English into the original language that was used by the customer.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))

if (params.payload.output.generic[0].text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnnn'
       },
  headers: {
    "Content-Type":"text/plain"
},
  body: [
        params.payload.context.skills['actions skill'].user_defined.original_input
  ],
  json: true,
};
    return rp(options)
   .then(res => {
     //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
       params.payload.context.skills['actions skill'].user_defined['language'] = res.languages[0].language;
       console.log(JSON.stringify(params))
       return params;
})
}
else {
   params.payload.context.skills['actions skill'].user_defined['language'] = 'none';
   return param
}
};
```

The second web action in the sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
   if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
```

```
  const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
  auth: {
        'username': 'apikey',
        'password': 'nnn'
     },
  body: {
     text: [
        params.payload.output.generic[0].text
        ],
        target: params.payload.context.skills["actions skill"].user_defined.language
  },
  json: true
  };
    return rp(options)
    .then(res => {
       params.payload.context.skills["actions skill"].user_defined["original_output"] = params.payload.output.generic[0].text;
       params.payload.output.generic[0].text = res.translations[0].translation;
       return {
         body : params
       }
})
}
  return {
    body : params
  }
};
```

Removing the webhook

If you do not want to process message responses with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to remove the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Post-message webhook**.
4. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Post-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Calling a service after processing a message for IBM Cloud Pak for Data - classic experience

Use a post-message webhook to call an external service after your assistant generates a response.

You can use post-message webhooks for the following use cases:

- Retrieve a response from an external source by using custom action IDs.
- Translate the assistant's response to the user's language.
- Reinsert personal data that was removed earlier for privacy.

Learn more

For more information about related features and details, see the following resources:

- Making a call before processing a message
- Calling a service before processing a message for IBM Cloud Pak for Data - classic experience
- Making a programmatic call from dialog
- API reference

Before you begin

Your webhook service must meet these technical requirements:

- Do not set up or test the webhook in a production environment.
- The call must be a POST HTTP request.

- The request and response must use JSON (Content-Type: application/json).
- The response must return within 30 seconds.

Procedure

This section covers the procedure to define, test, and remove post-message webhooks for Cloud Pak for Data - classic experience.

Webhook configuration

To add the webhook details, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.
2. Click **Webhooks** > **Post-message webhook**.
3. Set the **Post-message webhook** switch to **Enabled**.
4. In the **Synchronous event**, select from one of the following options:
   - Continue processing user input without webhook update if there is an error.
   - Return an error to the client if the webhook call fails.

   For more information, see Configuring webhook error handling for postprocessing.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, maybe you store your assistant's responses in a separate content management system. When the assistant understands the input, the processed action returns a unique ID that corresponds to a response in your CMS. To call a service that retrieves a response from your CMS for a given unique ID, specify the URL for your service instance. For example, `https://example.com/get_answer`.

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. Fill in the **Secret field**. For more information, see Adding a secret.
7. In the **Timeout** field, specify the time duration (in seconds) that you want the assistant to wait for a response from the webhook before it returns an error. The timeout duration cannot be shorter than 1 second or longer than 30 seconds.
8. In the **Headers** section, click **Add header +** to add any headers that you want to pass to the service, one at a time.

   Note: The service automatically sends an Authorization header with a JWT. If you want to handle authorization yourself, add your own authorization header and the service uses it instead.

   If the external application that you call returns a response, it might be able to send a response in different formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to indicate that you want the resulting value to be returned in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

9. After you save the header value, the string is replaced by asterisks and can't be viewed again.

10. Your webhook details are saved automatically.

Adding a secret

Add a client secret in the **Secret** field to pass with the request for authentication with the external service:

- Enter the key as a text string, such as `purple unicorn`.
- Use a maximum of 1,024 characters.
- Do not use context variables.

The external service is responsible for checking and verifying the secret. If no token is required, specify any string. This field cannot be left empty.

> **Note:** To view the secret as you enter it, click the **Show password** icon before typing. After saving the secret, asterisks replace the string, and you can't view it again.

For more information about how this field is used, see Webhook security.

Webhook security

Authenticate the webhook request by verifying the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call:

- For new webhooks or webhooks updated through **Edit authentication**, the authorization header is ignored.
- For existing webhooks with a saved authentication header, the **Edit authentication** button is disabled.
- Updating an existing webhook to use the new authentication configuration changes its behavior.

If you need to test the JWT verification, you can add code to the external service. For example, if you specify `purple unicorn` in the **Secret** field, you can use the following code:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

Configuring webhook error handling for postprocessing

You can decide whether an error returns in the postprocessing step if the webhook call fails. You have two options:

- **Continue processing user input without webhook update if there is an error** : The assistant ignores the errors and processes the message without the webhook result. If postprocessing is useful but not essential, consider this option.
- **Return an error to the client if the webhook call fails** : If postprocessing is critical after the assistant sends a response, select this option.

> ⚠ **Important:** When you enable **Return an error to the client if the webhook call fails** , everything stops until the postprocessing step is completed successfully.

Regularly test the external process to identify potential failures. If necessary, adjust this setting to prevent disruptions in the response processing.

Testing the webhook

> ⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is used in a production environment.

The webhook is triggered only when your assistant processes a message and a response is ready to be returned to the channel.

Troubleshooting the webhook

The following error codes can help you track down the cause of issues you might encounter. If you have a web chat integration, for example, you know that your webhook has an issue if every test message you submit returns a message such as `There is an error with the message you just sent, but feel free to ask me something else`. If this message is displayed, use a REST API tool, such as cURL, to send a test `/message` API request, so you can see the error code and the

full message that is returned.

Error code and message	Description
422 Webhook responded with invalid JSON body	The webhook's HTTP response body could not be parsed as JSON.
422 Webhook responded with `[500]` status code	A problem occurred with the external service that you called. The code failed or the external server refused the request.
500 Processor Exception : `[connections to all backends failing]`	An error occurred in the webhook microservice. It could not connect to backend services.

Error code details

Example request body

It is useful to know the format of the request post-message webhook body so that your external code can process it.

The payload contains the response body that your assistant returns for the version 2 of the `/message`, stateful and stateless, API call. The event name `message_processed` indicates that the post-message webhook generates the request. For more information about the message request body, see the API reference.

The following sample shows how a simple request body is formatted:

```
{
 "event": {
    "name": "message_processed"
},
"options": {},
"payload": {
    "output": {
        "intents": [
            {
                "intent": "General_Greetings",
                "confidence": 1
            }
        ],
        "entities": [],
        "generic": [
            {
                "response_type": "text",
                "text": "Hello. Good evening"
            }
        ]
    },
    "user_id": "test user",
    "context": {
        "global": {
            "system": {
                "user_id": "test user",
                "turn_count": 11
            },
            "session_id": "sxxx"
        },
        "skills": {
            "actions skill": {
                "user_defined": {
                    "var": "anthony"
                },
                "system": {
                    "state": "nnn"
                }
            }
        }
    }
}
```

Example 1

This example shows how to add `y'all` to the end of each response from the assistant.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The post-message webhook calls an IBM Cloud Functions web action name `add_southern_charm`.

The node.js code in the `add_southern_charm` web action looks as follows:

```
function main(params) {
  console.log(JSON.stringify(params))
  if (params.payload.output.generic[0].text !== '') {
      //Get the length of the input text
        var length = params.payload.output.generic[0].text.length;
        //create a substring that removes the last character from the input string, which is typically punctuation.
        var revision = params.payload.output.generic[0].text.substring(0,length-1);
      const response = {
         body : {
            payload : {
               output : {
                  generic : [
                     {
                        //Replace the input text with your shortened revision and append y'all to it.
                       "response_type": "text",
                       "text": revision + ', ' + 'y\'all.'
                     }
                  ],
               },
            },
         },
      };
      return response;
  }
  else {
    return {
       body : params
    }
  }
}
```

Example 2

This example shows how to translate a message response back to the customer's language. It works only if you perform the steps in Example 2 to define a pre-message webhook that translates the original message into English.

Define a sequence of web actions in IBM Cloud Functions. The first action in the sequence checks for the language of the original incoming text, which you stored in a context variable named `original_input` in the pre-message webhook code. The second action in the sequence translates the dialog response text from English into the original language that was used by the customer.

In the post-message webhook configuration page, the following values are specified:

- **URL**: `https://your-webhook-url/`
- **Secret**: None
- **Header name**: Content-Type
- **Header value**: application/json

The node.js code for the first web action in your sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
console.log(JSON.stringify(params))
```

```
if (params.payload.output.generic[0].text !== '') {
const options = { method: 'POST',
  url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/identify?version=2018-05-01',
  auth: {
          'username': 'apikey',
          'password': 'nnnn'
       },
  headers: {
    "Content-Type":"text/plain"
},
  body: [
         params.payload.context.skills['actions skill'].user_defined.original_input
  ],
  json: true,
};
     return rp(options)
    .then(res => {
      //Set the language property of the incoming message to the language that was identified by Watson Language Translator.
        params.payload.context.skills['actions skill'].user_defined['language'] = res.languages[0].language;
        console.log(JSON.stringify(params))
        return params;
})
}
else {
    params.payload.context.skills['actions skill'].user_defined['language'] = 'none';
    return param
}
};
```

The second web action in the sequence looks as follows:

```
let rp = require("request-promise");

function main(params) {
  console.log(JSON.stringify(params))
    if ((params.payload.context.skills["actions skill"].user_defined.language !== 'en') && (params.payload.context.skills["actions skill"].user_defined.language !== 'none')) {
    const options = { method: 'POST',
    url: 'https://api.us-south.language-translator.watson.cloud.ibm.com/instances/572b37be-09f4-4704-b693-3bc63869nnnn/v3/translate?version=2018-05-01',
    auth: {
            'username': 'apikey',
            'password': 'nnn'
         },
    body: {
        text: [
            params.payload.output.generic[0].text
            ],
            target: params.payload.context.skills["actions skill"].user_defined.language
    },
    json: true
    };
      return rp(options)
      .then(res => {
          params.payload.context.skills["actions skill"].user_defined["original_output"] = params.payload.output.generic[0].text;
          params.payload.output.generic[0].text = res.translations[0].translation;
          return {
            body : params
          }
  })
  }
  return {
    body : params
  }
};
```

Removing the webhook

If you do not want to process message responses with a webhook, complete the following steps:

1. For the assistant that you want to configure, click the ⋮ icon, and then choose **Settings**.

2. Click **Webhooks** > **Post-message webhook**.
3. Do one of the following steps:

- To stop calling a webhook to process every incoming message, set the **Post-message webhook** switch to **Disabled**.
- To change the webhook that you want to call, click **Delete webhook**.

Logging activity with a webhook

Plus

You can log activity by making a call to an external service or application every time a customer submits input to the assistant.

A webhook is a mechanism that you can use to call out to an external program based on events in your program.

Note: This feature is available only to Plus and Enterprise plan users. The Plus plan allows no more than 5 log webhooks per instance. This limit does not apply to Enterprise plan instances.

Add a log webhook to your assistant if you want to use an external service to log activity. You can log two kinds of activity:

- **Messages and responses**: The log webhook is triggered each time that the assistant responds to customer input. You can use this option as an alternative to the built-in analytics feature to handle logging yourself. (For more information about the built-in analytics support, see Review your entire assistant at a glance.)

  Important: If you are using a custom channel, the log webhook works with the v2 `/message` API only (stateless and stateful). For more information, see the API reference. All built-in channel integrations use this API.

- **Call detail records (CDRs)**: The log webhook is triggered after each telephone call a user makes to your assistant that uses the phone integration. A Call Detail Record (CDR) is a summary report that documents the details of a telephone call, including phone numbers, call length, latency, and other diagnostic information. CDR records are only for assistants that use the phone integration.

The log webhook does not return anything to your assistant.

Note: For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

Defining the webhook

You can define one webhook URL to use for logging every incoming message or CDR event.

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.

To add the webhook details, complete the following steps:

1. In your assistant, open the environment where you want to configure the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Log webhook**.
   a. Or, if you're using the classic experience, open the **Assistants** page.
   b. For the assistant you want to configure, click the icon, and then choose **Settings**.
   c. Click **Webhooks**, then click **Log webhook**.
4. Set the **Log webhook** switch to **Enabled**.

   If you cannot enable the webhook, you might need to upgrade your service plan.
5. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts. For example, `https://example.com/my_log_service`.

   You must specify a URL that uses the SSL protocol, so specify a URL that begins with `https`.
6. In the **Secret** field, add a token to pass with the request that can be used to authenticate with the external service.

The secret must be specified as a text string, such as `purple unicorn`. The maximum length is 1,024 characters. You cannot specify a context variable.

It is the responsibility of the external service to check for and verify the secret. If the external service does not require a token, specify any string that you want. You cannot leave this field empty.

> **Note:** If you want to see the secret as you enter it, click the **Show password** icon before you start typing. After you save the secret, the string is replaced by asterisks and can't be viewed again.

7. Click the appropriate checkboxes to select which kinds of activity you want to log:
   - To log messages and responses, select **Subscribe to conversation logs**.
   - To log CDR events for the phone integration, select **Subscribe to CDR (Call Detail Records)**.
8. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

   The service automatically sends an `Authorization` header with a JWT; you do not need to add one. If you want to handle authorization yourself, add your own authorization header and it is used instead.

   > **Note:** After you save the header value, the string is replaced by asterisks and can't be viewed again.

Your webhook details are saved automatically.

Removing the webhook

If you decide you do not want to log messages with a webhook, complete the following steps:

1. In your assistant, go to **Environments** and open the environment where you want to configure the webhook.
2. Click the icon to open the environment settings.
3. On the **Environment settings** page, click **Log webhook**.
   a. Or, if you're using the classic experience, open the **Assistants** page.
   b. For the assistant you want to configure, click the icon, and then choose **Settings**.
   c. Click **Webhooks**, then click **Log webhook**.
4. Do one of the following things:
   - To change the webhook that you want to call, click **Delete webhook** to delete the currently specified URL and secret. You can then add a URL and other details.
   - To stop calling a webhook to log every message and response, click the *Log webhook* switch to disable the webhook altogether.

Webhook security

To authenticate the webhook request, verify the JSON Web Token (JWT) that is sent with the request. The webhook microservice automatically generates a JWT and sends it in the `Authorization` header with each webhook call. It is your responsibility to add code to the external service that verifies the JWT.

For example, if you specify `purple unicorn` in the **Secret** field, you might add code such as:

```
const jwt = require('jsonwebtoken');
...
const token = request.headers.authentication; // grab the "Authentication" header
try {
  const decoded = jwt.verify(token, 'purple unicorn');
} catch(err) {
  // error thrown if token is invalid
}
```

Webhook request body

The request body that the webhook sends to the external service is a JSON object with the following structure:

```
{
  "event": {
    "name": "{event_type}"
```

```
  },
  "payload": {
    ...
  }
}
```

where `{event_type}` is either `message_logged` (for messages and responses) or `cdr_logged` (for CDR events).

The `payload` object contains the event data to be logged. The structure of the `payload` object depends on the type of event.

Message event payload

For `message_logged` events, the `payload` object contains data about a message request that is sent to the assistant and the message response that is returned to the integration or client application. For more information about the fields that are part of message requests and responses, see the API reference.

> ⚠ **Important:** The log webhook payload might include data that is not currently supported by the API. Any fields that are not defined in the API reference documentation are subject to change.

CDR event payload

For `cdr_logged` events, the `payload` object contains data about a Call Detail Record (CDR) event that was handled by the phone integration. The structure of the `payload` object for a CDR event is as shown by this example:


```
{
  "primary_phone_number": "+18005550123",
  "global_session_id": "9caa8bad-aaa8-4a5a-a4b5-62bccc703d15",
  "failure_occurred": false,
  "transfer_occurred": false,
  "active_calls": 0,
  "warnings_and_errors": [
    {
      "code": "CWSMR0033W",
      "message": "CWSMR0033W: The inbound RTP audio stream jitter of 43 ms exceeds the maximum jitter threshold of 30 ms."
    },
    {
      "code": "CWSMR0070W",
      "message": "CWSMR0070W: A request to the Watson Speech To Text service failed for the following reason = Unexpected server response: 403, response headers = {\"strict-transport-security\":\"max-age=31536000; includeSubDomains;\",\"content-length\":\"157\",\"content-type\":\"application/json\",\"x-dp-watson-tran-id\":\"23860083-88b6-41d7-9130-30bbfebe647e\",\"x-request-id\":\"23860083-88b6-41d7-9130-30bbfebe647e\",\"x-global-transaction-id\":\"6c764df3-81db-41bb-a14f-62384facffca\",\"server\":\"watson-gateway\",\"x-edgeconnect-midmile-rtt\":\"1\",\"x-edgeconnect-origin-mex-latency\":\"28\",\"date\":\"Thu, 13 May 2021 20:31:12 GMT\",\"connection\":\"keep-alive\"}, response body = {\"code\":403,\"trace\":\"23860083-88b6-41d7-9130-30bbfebe647e\",\"error\":\"Forbidden\",\"more_info\":\"[https://cloud.ibm.com/docs/watson?topic=watson-forbidden-error](https://cloud.ibm.com/docs/watson?topic=watson-forbidden-error)\"}, x-global-transaction-id = 6c764df3-81db-41bb-a14f-62384facffca. The Media Relay will reattempt to send the request."
    }
  ],
"realtime_transport_network_summary": {
  "inbound_stream": {
    "average_jitter": 4,
    "canonical_name": "b74f3689-1ae8-4a0a-bde3-adf5b488553e",
    "maximum_jitter": 18,
    "packets_lost": 0,
    "packets_transmitted": 952,
    "tool_name": ""
  },
  "outbound_stream": {
    "average_jitter": 0,
    "canonical_name": "voice.gateway",
    "maximum_jitter": 0,
    "packets_lost": 0,
    "packets_transmitted": 838,
    "tool_name": "IBM Voice Gateway/1.0.7.0"
  }
},
"call": {
  "start_timestamp": "2021-10-12T20:54:02.591Z",
  "stop_timestamp": "2021-10-12T20:54:20.375Z",
  "milliseconds_elapsed": 17784,
  "outbound": false,
```

```
  "end_reason": "assistant_hangup",
  "security": {
    "media_encrypted": false,
    "signaling_encrypted": false,
    "sip_authenticated": false
  }
},
"session_initiation_protocol": {
  "invite_arrival_time": "2021-10-12T20:54:00.565Z",
  "setup_milliseconds": 2026,
  "headers": {
    "call_id": "17465345_115257202@10.90.150.99",
    "from_uri": "sip:+18885550456@pstn.twilio.com",
    "to_uri": "sip:+18005550123@public.voip.us-south.assistant.test.watson.cloud.ibm.com"
  }
},
"max_response_milliseconds": {
  "assistant": 339,
  "text_to_speech": 535,
  "speech_to_text": 0
},
"assistant_interaction_summaries": [
{
  "session_id": "7874ec3a-1330-4180-afe1-46bfb220af5b",
  "assistant_id": "97f16ba4-ad94-41af-aa6c-33cd56ad5e7e",
  "turns": [
    {
      "assistant": {
        "log_id": "58bebfd1-0118-419b-a555-b152a1efbbe8",
        "response_milliseconds": 339,
        "start_timestamp": "2021-10-12T20:54:00.722Z"
      },
      "request": {
        "type": "start"
      },
      "response": [
        {
          "barge_in_occurred": true,
          "streaming_statistics": {
            "response_milliseconds": 301,
            "start_timestamp": "2021-10-12T20:54:00.722Z",
            "stop_timestamp": "2021-10-12T20:54:01.023Z",
            "transaction_id": "3dce431c-fb2f-4b62-9fce-585f4e06fe00"
          },
          "type": "text_to_speech"
        }
      ]
    },
    {
"assistant": {
  "log_id": "38f36bfb-c2aa-4600-9418-6ab422664e31",
  "response_milliseconds": 158,
  "start_timestamp": "2021-10-12T20:54:05.621Z"
},
"request": {
  "type": "dtmf"
},
"response": [
  {
    "type": "disable_speech_barge_in"
  },
  {
    "type": "text_to_speech",
    "barge_in_occurred": false,
    "streaming_statistics": {
      "transaction_id": "af4c47c3-5cc4-43c8-9b9c-81d6f997c52f",
      "start_timestamp": "2021-10-12T20:54:06.321Z",
      "stop_timestamp": "2021-10-12T20:54:14.338Z",
      "response_milliseconds": 535
    }
  },
```

```
      {
        "type": "enable_speech_barge_in"
      },
      {
        "type": "text_to_speech",
        "barge_in_occurred": true,
        "streaming_statistics": {
          "transaction_id": "eafdd846-2829-4e1a-8068-b1035510b1e1",
          "start_timestamp": "2021-10-12T20:54:14.795Z",
          "stop_timestamp": "2021-10-12T20:54:20.388Z",
          "response_milliseconds": 447
        }
      }
    ]
  },
  {
    "assistant": {
      "log_id": "07d74b35-0205-43e4-923c-1e43e1cb429c",
      "response_milliseconds": 0,
      "start_timestamp": "2021-10-12T20:54:20.377Z"
    },
    "request": {
      "type": "hangup"
        },
        "response": []
      }
    ]
  }
]
}
```


For more information about the structure of the CDR event payload, see CDR log event reference.

Defining the authentication method for pre-message and post-message webhooks

This document shows the process for configuring authentication for pre-message webhooks and post-message webhooks in watsonx Assistant. It covers the available authentication methods and how to set them up.

Overview

Webhooks allow external systems to communicate with watsonx Assistant. Authentication ensures that only authorized sources can trigger webhooks. This documentation describes the process for configuring and managing webhook authentication, which introduces an updated method for authenticating webhooks.

Before you begin

Before you configure the webhook authentication:

- You must have write permissions in the environment.
- You must have authentication details of the target server, including token request URLs (if needed) and any secrets, such as a password or token.

Procedure

1. Go to **Home** > **Environments**.
2. Select **Settings** from either the **Draft** tab > **Draft environment** or the **Live** tab > **Live environment**.
3. Select from either **Pre-message webhook** or **Post-message webhook**, according to what you want to define.
4. Scroll down to **Webhook setup**, and paste the API URL.
5. Click **Edit authentication** to open the **Authentication set up** page.
6. In the dropdown, choose one of the following options:
    - No authentication
    - Basic auth

    - Bearer auth
    - API key auth
    - OAuth 2.0
    - Signed JWT
7. Click **Save**.

No authentication

This is the default option.

Basic auth

1. Enter a username and password.

Bearer auth

1. Enter the bearer token.

API key auth

1. Enter the **API key name** and **API key**.

Signed JWT

1. Enter the **Secret**.
2. Click the **Show password** icon to view the secret.

OAuth 2.0

> **Note:** If you use the **Scope string**, it must be a space-delimited set of one or more authentication scopes defined by the target server. For example, write, read+write, email-read, and so on.

1. In **Grant type**, choose one of the following options:
    - Password
    - Client credentials
    - Authorization code
    - Custom
2. Click **Save**.

Password

1. Enter the **Username** of your webhook.
2. Enter the **Password** for your webhook service.
3. Enter the **Client ID** for your webhook authentication service.
4. Enter the **Client secret** to authenticate your webhook.
5. Enter the **Token URL**.
6. Enter the **Refresh token URL**.
7. **Optional:** If your service needs a scope string, enter the **Scope string** as defined by the target server.
8. In **Client authentication**, you must choose one of the following options:
    - **Send as Basic Auth header:** Authentication credentials will be sent in the HTTP header.
    - **Send as Body:** Authentication credentials will be sent in the request body.
9. Enter the **Header prefix**, for example: Bearer.

Client credentials

1. Enter the **Client ID** for your webhook authentication service.
2. Enter the **Client secret** to authenticate your webhook.
3. Enter the **Token URL**.
4. Enter the **Refresh token URL**.
5. **Optional:** If your service needs a scope string, enter the **Scope string** as defined by the target server.
6. In **Client authentication**, you must choose one of the following options:
    - **Send as Basic Auth header:** Authentication credentials will be sent in the HTTP header.
    - **Send as Body:** Authentication credentials will be sent in the request body.
7. Enter the **Header prefix**, for example: Bearer.

Authorization code

1. Enter the **Client ID** for your webhook authentication service.
2. Enter the **Authorizing server URL**.
3. Enter the **Token URL**.
4. Enter the **Refresh token URL**.
5. **Optional:** If your service needs a scope string, enter the **Scope string** as defined by the target server.
6. In **Client authentication**, you must choose one of the following options:
    - **Send as Basic Auth header:** Authentication credentials will be sent in the HTTP header.
    - **Send as Body:** Authentication credentials will be sent in the request body.
7. Enter the **Header prefix**, for example: Bearer.
8. **Optional:** Depending on the target server, copy the Redirect url to your OAuth app's 'Callback URL' field.
9. Click **Grant Access**.
10. Complete the steps on the page that presents by the granting server.
11. You are redirected back to the Assistant, and the edit modal re-opens.
12. Enter the **Client secret** under **Client ID** now that the field is visible.

Custom

1. Enter the **Custom grant type name** of your webhook.
2. Enter the **Token URL**.
3. Enter the **Refresh token URL**.
4. **Optional:** If your service needs a scope string, enter the **Scope string** as defined by the target server.
5. In **Client authentication**, you must choose one of the following options:
    - **Send as Basic Auth header:** Authentication credentials will be sent in the HTTP header.
    - **Send as Body:** Authentication credentials will be sent in the request body.
6. Enter the **Header prefix**, for example: Bearer.

If you need to add custom secrets to your application, follow these steps:

1. Click **Add secret +**.
2. Type the **Secret name** and the **Secret value**.
3. **Optional:** If you want to add more secret names and secret values, click **Add secret +**.
4. Click **Add parameter +**.

5. Type the **Parameter name**, and the **Parameter value**.
6. **Optional:** If you want to add more parameter names and parameter values, click **Add parameter +**.

Defining an audio webhook

Note: This feature is currently supported only with the Genesys Audio Connector of the Phone Integration.

An audio webhook calls an external service or application whenever a record response type is used to collect audio. The external service processes the audio, and if an error occurs, it disconnects the call. The audio webhook is often used to store the audio for compliance reasons, as watsonx Assistant does not store the audio.

Before you begin

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.
- The request body must be a JSON object (Content-Type: multipart/form-data).
- The call must return within 30 seconds or less.

For more information on recording audio, see Recording a caller's utterance.

Procedure

To add the webhook details, complete the following steps:

1. Go to **Home** > **Environments**.
2. Click **Settings** from either the **Draft** tab > **Draft environment** or the **Live** tab > **Live environment**.
3. Click **Audio webhook**.
4. Set the **Audio webhook** switch to `Enabled`.
5. In the **URL** field, add the URL for the external application to which you want to send the HTTP POST request. Ensure that the URL uses the SSL protocol (for example, starting with https).
6. In the **Secret** field, add a private key to pass with the request to authenticate with the external service. The key must be specified as a text string (for example, purple unicorn), with a maximum length of 1,024 characters.

   You cannot specify a context variable. If the external service does not require a token, specify any string that you want. You cannot leave this field empty. To view the secret as you enter it, click **Show password** before typing. After saving the secret, asterisks replace the string, and you cannot view it again. For more information about how this field is used, see Webhook security for the classic experience only.
7. In the **Timeout** field, specify the duration (in seconds) that you want the assistant to wait for a response from the webhook before returning an error. The timeout duration must be between 1 and 30 seconds.
8. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

   For example, if the external application that you call returns a response, it might be able to send a response in multiple formats. The webhook requires that the response is formatted in JSON. The following table illustrates how to add a header to ensure that the resulting value to be returned is in JSON format.

Header name	Header value
`Content-Type`	`application/json`

Header example

After you save the header value, the string is replaced by asterisks and can't be viewed again.

Your webhook details are saved automatically.

For more examples, see audio webhook examples.

Testing the webhook

⚠ **Important:** Do extensive testing of your webhook before you enable it for an assistant that is being used in a production environment.

The webhook is triggered when a recording response type is used during a phone call. If the request to the webhook fails, the call disconnects. You can look at the phone integration logs for more details as to why the webhook failed. For more information, see Phone integration troubleshooting.

Request body

It is useful to know the format of the request body of the audio webhook so that your external code can process it.

The payload contains the audio and metadata as Content-Type: multipart/form-data. An example of the request is:

```
POST /audio-webhook HTTP/1.1
Content-Type: multipart/form-data; boundary=----------3676416B-9AD6-440C-B3C8-FC66DDC7DB45
----------3676416B-9AD6-440C-B3C8-FC66DDC7DB45
Content-Disposition: form-data; name="metadata"
Content-Type: application/json
{
   "assistant_id": "dadf4b56-3b67-411a-b48d-079806b626d3",
   "environment_id": "6205aead-fe91-44af-bfe1-b4435015ba23",
   "session_id": "50989a59-9976-4b3f-9a98-af42adcad69a",
   "recording_id": "3daeb5d2-f52b-4c3e-a869-328b6fc6327c",
   "start_timestamp": "2024-10-21T17:22:07.789Z",
   "stop_timestamp": "2024-10-21T17:22:37.789Z"
}
----------3676416B-9AD6-440C-B3C8-FC66DDC7DB45
Content-Disposition: form-data; name="audio_recording"
Content-Type: audio/mulaw;rate=8000

<binary data>
```

Recording a caller's utterance

Note: This feature is currently supported only with the Genesys Audio Connector of the Phone Integration.

The **Record** feature captures the caller's utterance to verify their identity, provide consent, or authorize actions. You can then play it back to the caller for confirmation to proceed with the action.

Before you begin

Go to Defining an audio webhook to collect and save audio from the user.

Adding a recording

Add a record response to collect the user's voice and play it back to confirm a transaction.

To add the record response type, do the following steps:

1. Go to **Home** > **Actions**.
2. Select your **Action** name or create a **New action +** > **Conversation steps**.
3. In the **Assistant says** field, click the **Add recording** icon .
4. In the **Customer prompt** field, type the message that you want the caller to hear before the recording starts. For example, "Please state your name to authorize the payment. Press pound to capture the recording."
5. You can choose to play a tone before the recording starts. In the **Input method** field, select **Beep** to play the tone, otherwise select **Silent (pause)**.
6. Select the keypad **End recording key** that the caller needs to press to end the recording. The default value is **#**.
7. In the **Playback confirmation** field, type the message that the caller hears after the recording completes to confirm the recording. The placeholder variable **[recording]** is used where the caller listens to the audio recording.
8. Click **Apply**.

Making a programmatic call from dialog

To make a programmatic call, define a webhook that sends a POST request callout to an external application that performs a programmatic function. You can then start the webhook from one or more dialog nodes.

Note: If you are using actions instead of dialog, you can use a custom extension to make programmatic calls. For more information, see Calling a custom extension.

A webhook is a mechanism that you can use to call out to an external program based on an event in your program. When used in a dialog, a webhook is triggered when the assistant processes a node with a webhook that is enabled. The webhook collects data that you specify or that you collect from the user during the conversation and save in context variables. It sends the data as part of an HTTP POST request to the URL that you specify as part of your webhook definition. The URL that receives the webhook is the listener. It performs a predefined action that uses the information that you pass to it as specified in the webhook definition, and can optionally return a response.

You can use a webhook to do the following types of things:

- Validate information that you collected from the user.
- Interact with an external web service to get information. For example, you might check on the expected arrival time for a flight from an air traffic service or get a forecast from a weather service.
- Send requests to an external application, such as a restaurant reservation site, to complete a simple transaction on the user's behalf.
- Trigger an SMS notification.

For information about how to call a client application, see Requesting client actions.

Note: For environments where private endpoints are in use, keep in mind that a webhook sends traffic over the internet.

Defining the webhook

You can define one webhook URL for a dialog, and then call the webhook from one or more dialog nodes.

The programmatic call to the external service must meet these requirements:

- The call must be a POST HTTP request.
- The request body must be a JSON object (`Content-Type: application/json`).
- The response must be a JSON object (`Accept: application/json`).
- The call must return in 8 seconds or less. If started more than once in a single message call through dialog nodes, all of those invocations must return in 8 seconds or less.

Tip: If your external service supports GET requests only, or if you need to specify URL parameters dynamically at run time, consider creating an intermediate service that accepts a POST request with a JSON payload that contains any runtime values. The intermediate service can then make a request to the target service, passing these values as URL parameters, and then return the response to the dialog.

Tip: If you need to call a service that might not return within 8 seconds, you can manage the call through a custom client application and pass the information to the dialog as a separate step. For more information, see Requesting client actions.

To add the webhook details, complete the following steps:

1. In the dialog where you want to add the webhook, click **Webhooks**.
2. In the **URL** field, add the URL for the external application to which you want to send HTTP POST request callouts.

   For example, to call the Language Translator service, specify the URL for your service instance.

   ```
   https://api.us-south.language-translator.watson.cloud.ibm.com/v3/translate?version=2018-05-01
   ```

   If the external application that you call returns a response, it must be able to send back a response in JSON format. For the Language Translator service, for example, you must specify the format in which you want the result to be returned. You can do so by passing a header to the service.
3. In the Headers section, add any headers that you want to pass to the service one at a time by clicking **Add header**.

   For example, this header indicates that the request is in JSON format.

Header name	Header value

Content-Type	application/json

Header example

4. If the external service requires that you pass basic authentication credentials with the request, then provide them. Click **Add authorization**, add your credentials to the **User name** and **Password** fields, and then click **Save**.

   The product creates a base-64 encoded ASCII string from the credentials and generates a header that it adds to the page for you.

Header name	Header value
Authorization	Basic <encoded-credentials>

Header example

> **Tip:** If you use the web chat integration and enable security, you can use the same token that you use to secure the web chat in the Authorization header. For more information, see Web chat: Reusing the JWT for webhook authentication .

Your webhook details are saved automatically.

Adding a webhook callout to a dialog node

To use a webhook from a dialog node, you must enable webhooks on the node, and then add details for the callout.

1. Find the dialog node where you want to add a callout. The callout to the webhook occurs whenever this node is triggered during a conversation with a user.

   For example, you might want to send a callout to the webhook from the #General_Greetings node.

2. Click to open the dialog node, and then click **Customize**.
3. Scroll down to the webhook section. Set the **Call out to webhooks / actions** switch to **On**.
4. Select **Call a webhook**, and then click **Apply**.

> **Note:** If you did not have it enabled already, the **Multiple conditioned responses** switch is set to **On** automatically and you cannot disable it. This setting is enabled to support adding different responses depending on the success or failure of the webhook call. If you have a response that is specified for the node already, it becomes the first conditional response.

5. Add any data that you want to pass to the external application as key and value pairs in the *Parameters* section.

> **Note:** Parameters are passed as request body properties. You cannot specify query parameters or URL parameters in a dialog node. These parameters can be configured with static values only as part of the webhook definition. For more information, see Defining the webhook.

For example, if you call the Language Translator service, you must provide values for the following parameters:

Key	Value	Description
model_id	en-es	Identifies the input and output languages. In this example, the request is for text in English (en) to be translated into Spanish (es).
text	How are you?	This parameter contains the text string that you want the service to translate. You can hardcode this value, pass a context variable, such as $saved_text, or pass the user input to the service directly, by specifying <? input.text ?> as this value.

Parameter example

In more complex use cases, you might collect information during a conversation with a user about their travel plans, for example. You can collect dates and destination information and save it in context variables that you can pass to an external application as parameters.

Key	Value
depart_date	$departure

arrive_date	$arrival
origin	$origin
destination	$destination

Travel parameters example

6. Any response that is made by the callout is saved to the return variable. You can rename the variable that is automatically added to the **Return variable** field for you. If the callout results in an error, this variable is set to `null`.

   The generated variable name has the syntax `webhook_result_n`, where the `_n` suffix is incremented each time that you add a webhook callout to a dialog node. This naming convention ensures that context variable names are unique across the dialog. If you change the name, be sure to use a unique name.

7. In the conditional responses section, two response conditions are added automatically, one response to show when the webhook callout is successful and a return variable is sent back. And one response to show when the callout fails. You can edit these responses, and add more conditional responses to the node.

   - If the callout returns a response, and you know the format of the JSON response, then you can edit the dialog node response to include only the section of the response that you want to share with users.

     For example, the Language Translator service returns an object like this:

     ```
     {
     "translations":[
       {"translation":"¿Cómo estás?"}
     ],
     "word_count":3,
     "character_count":12
     }
     ```

     Use a SpEL expression that extracts only the translated text value.

Condition	Response
$webhook_result_1	Your words in Spanish: .
anything_else	The call to the external application failed. Please try again later.

     Conditional responses example

     If you use the recommended format for the response and the translation response that is shown earlier is returned, the assistant's response to the user would be: `Your words in Spanish: ¿Cómo estás?`

   - If you want to provide a specific response if the callout returns an empty string, meaning the call is successful, but the value that is returned is an empty string, you can add a conditional response that has a condition with syntax like this:

     ```
     $webhook_result_1.size() == 0
     ```

8. When you are done, click the X to close the node. Your changes are automatically saved.

Testing webhooks

When you first add a webhook callout, it can be useful to see exactly what is returned in the response from the external application, the data, and its format. Add this expression as the text response for the successful callout conditional response: `$webhook_result_n` where `n` is the appropriate number for the webhook that you are testing.

This response returns the full body of the return variable, so you can see what the callout is sending back and decide what to share with the user. You can then use the methods that are documented in Expression language methods to extract only the information you care about from the response.

Test whether certain user inputs can generate errors in the callout, and build in ways to handle those situations. Errors that are generated by the external application are stored in `output.webhook_error.<result_variable>`. You can use a conditional response like this while you are testing to capture such errors:

Condition	Response

output.webhook_error	The callout generated this error: `<? output.webhook_error.webhook_result_1 ?>`

Conditional response example

For example, you might not be authenticating the request properly (401), or you might be trying to pass a parameter with a name that is already in use by the external application. Test the webhook to discover and fix these types of errors before you deploy the webhook.

Removing a webhook

If you decide you do not want to make a webhook call from a dialog node, open the node's *Customize* page, and then switch Webhooks **Off**.

The *Parameters* section and the **Return variable** field are removed from the dialog node editor. However, any conditional responses that were added for you or that you added yourself remain.

The *Multiple conditioned responses* section is editable again. You can choose to switch off the feature. If you do so, then only the first conditional response is saved as the node's only text response.

To change the external service that you call from dialog nodes, edit the webhook details defined on the Webhooks page of the **Options** tab. If the new service expects different parameters to be passed to it, be sure to update any dialog nodes that call it.

Updating output.generic with a webhook

You can use a webhook to update `output.generic` and provide dynamic responses. For more information, see the blog article How to Dynamically Add Response Options to Dialog Nodes.

Migrating to watsonx Assistant

Administering your instance

Topics that cover tasks and areas in administering your instance of IBM® watsonx™ Assistant.

Topic	Description
Managing access	You can give other people access to your watsonx Assistant instance and resources, and control the level of access they get.
Managing your plan	A watsonx Assistant plan information reference and steps on upgrading your plan.
Activity log	Use the activity log to track changes. It gives you visibility into the modifications that are made to your assistant.
Activity tracking	As a security officer, auditor, or manager, you can use the activity tracking to see how users and applications interact with watsonx Assistant.
Securing your assistant	Data privacy, security, and governance solutions.
Backing up and restoring data	Back up and restore your data by downloading, and then uploading the data.
Deleting an instance of the service	You can delete an instance of the service from your account.
High availability and disaster recovery	watsonx Assistant is highly available within multiple IBM Cloud® locations (for example, Dallas and Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.
Failover options	Actions that you can take to increase the availability of watsonx Assistant for your organization.
Adding support for global audiences	watsonx Assistant supports individual features to varying degrees per language.
Switching between new and classic experiences	You can easily switch back and forth between watsonx Assistant and the classic experience.

Comparing actions and dialog

Choose the right type of conversation for your use case.

Actions benefits

Using actions is the best choice when you want to approach the assistant with a focus on content. Actions offers the following benefits:

- The process of creating a conversational flow is easier. People who have expertise with customer care can write the words that your assistant says. With a simplified process anyone can build a conversation. You don't need knowledge about machine learning or programming.
- Actions provide better visibility into the customer's interaction and satisfaction with the assistant. Because each task is discrete and has a clear beginning and ending, you can track user progress through a task and identify snags.
- The conversation designer doesn't need to manage data collected during the conversation. By default, your assistant collects and stores information for the duration of the current action. You don't need to take extra steps to delete saved data or reset the conversation. But if you want, you can store certain types of information, such as the customer's name, for the duration of a conversation.
- Many people can work at the same time in separate, self-contained actions. The order of actions within a conversation doesn't matter. Only the order of steps within an action matters. And the action author can use drag and drop to reorganize steps in the action for optimal flow.

Dialog benefits

A dialog-based conversation is the best choice when you want greater control over the logic of the flow. The dialog editor exposes more of the underlying artifacts (such as intents and entities) used to build the AI models. The dialog flow uses an if-then-else style structure that might be familiar to developers,

but not to content designers or customer-care experts.

How actions are different from dialog

If you are already familiar with dialog-based conversations, learn more about how actions compares.

Feature	Actions	Dialog
Automatic reset of context	✓	
Keep track of context	✓	✓
Collect info, as with slots	✓	✓
Contextual entities		✓
Collect numbers (@sys-number detection)	✓	✓
Detection of other system entities		✓
Connect to agent response type	✓	✓
Free text response type	✓	✓
Image response type	✓	✓
Options response type	✓	✓
Search skill response type	✓	✓
Rich text editor for text responses	✓	
User input validation	✓	✓
Step logic validation	✓	
Support multiple users by notifying them when simultaneous edits are made to the skill	✓	
Use SpEL expressions	✓	✓
Disambiguation	✓	✓
Digression support	✓	✓
Spelling correction	✓	✓
Webhook (before or after every message) support	✓	✓
Webhook (from a node) support	✓	✓
Webhook (log all messages) support	✓	✓

Conversational flow skill feature support

For some functions, there is parity but you follow different steps to implement the behavior you want.

- **Jump-to**: In actions, you can jump from one step to another. In a dialog, you use a jump-to to skip to a specific dialog node in the same branch of the conversation. With actions, you can jump to a different step within an action also. However, to do so, you use conditions on the intervening steps to prevent them from being processed, rather than using an explicit jump-to. The benefit of this approach is that it's easier to anticipate the path of a

conversation and follow it later if there are not multiple jump-tos sprinkled throughout the flow.

- **Slots**: In a dialog, you add slots to a dialog node to call out a set of values that you want to collect from the user, and that you will take and store in any order. In actions, every step in the action acts like a slot. If the user provides information that address step 10 when answering the question to step 1, both step 1 and step 10 are filled. In fact, if you want step 10 to ask the question explicitly, you must select the **Always ask for this** option on step 10.

Tip: *Want to get started with actions, but need features that are available from a dialog?* Use both. Dialog is your primary conversation with users, but you can call an action from your dialog to perform a discrete task. For more information, see Calling an actions from a dialog.

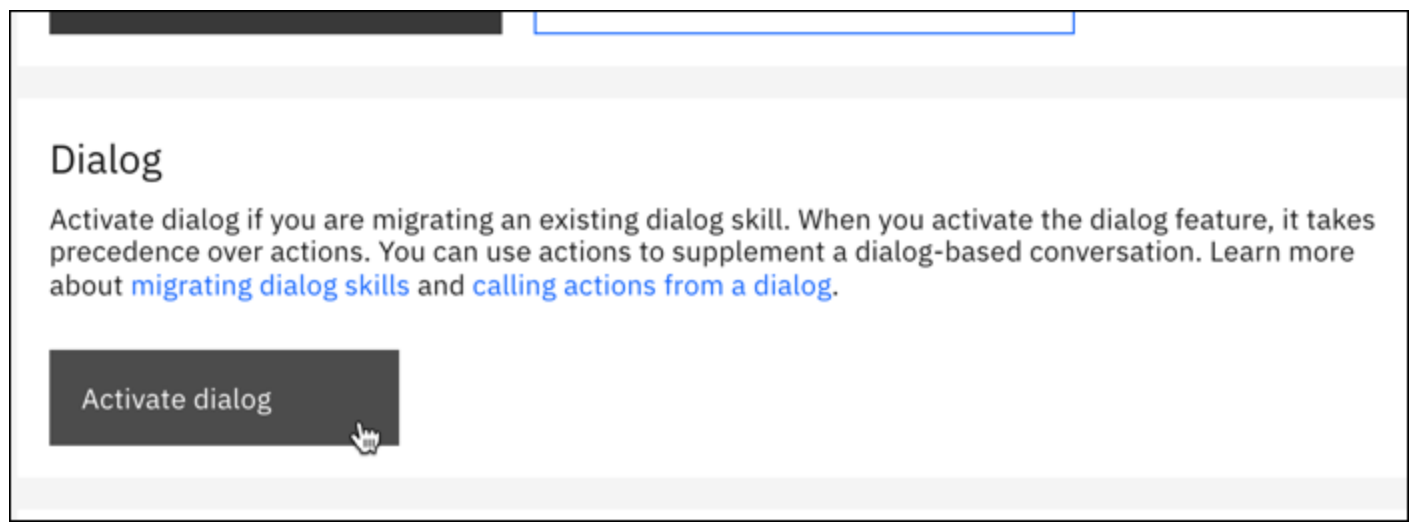
Activating dialog and migrating skills

To use an existing dialog skill, you need to activate the dialog feature in your assistant. Then you can upload your existing skill.

Activating dialog

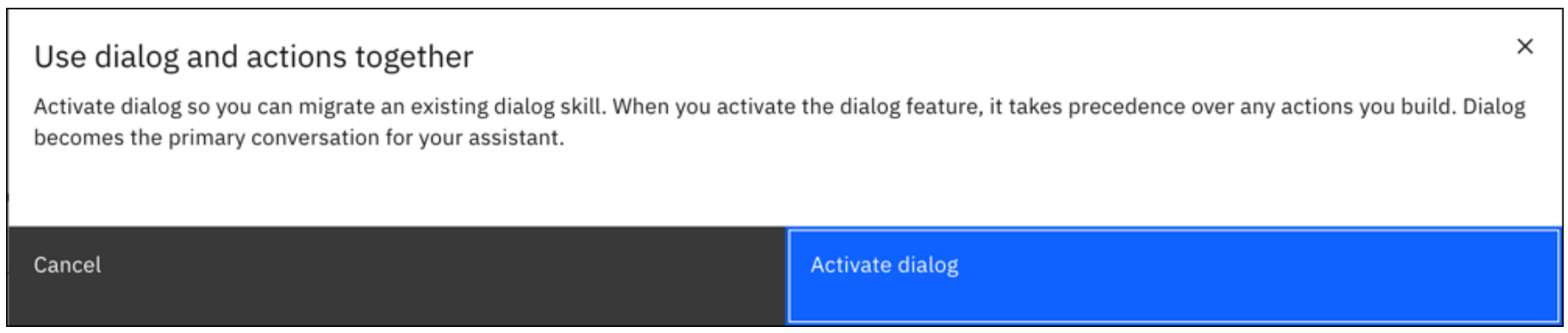
To activate the dialog feature:

1. Create or open an assistant where you want to use a dialog as the primary conversation with users.
2. Open **Assistant settings**.
3. Click **Activate dialog**.



Activate dialog

4. In the confirmation that displays, click **Activate dialog** again.



Activate dialog

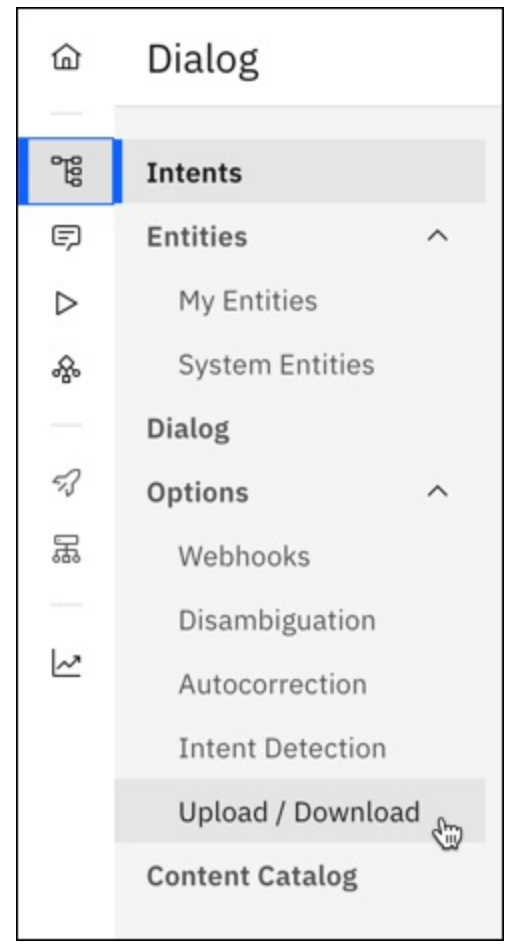
Once you activate the dialog feature, it takes precedence over actions. You can use actions to supplement a dialog-based conversation, but the dialog drives the conversation with users to match their requests. For more information, see Calling actions from a dialog.

Migrating existing skills

Once the dialog feature is activated, you can start a new dialog conversation from scratch if you want. But, you probably have an existing dialog skill that you want to migrate into watsonx Assistant.

To migrate an existing dialog skill:

1. Use the classic experience to download your dialog skill in JSON format.
2. In watsonx Assistant, open the **Dialog** page.
3. In **Options**, choose **Upload / Download**.



Upload dialog

4. On the **Upload** tab, upload the JSON file for your dialog skill.

Migrating intents and entities

You can migrate your intents and entities from the classic experience to watsonx Assistant.

Downloading intents

You can download intents to a CSV file, so you can then upload and reuse them as actions or example phrases in watsonx Assistant.

1. Go to the **Intents** page.
2. Select the intents that you want to download, and then click the **Download** icon .

   Important: If you plan to use a downloaded intent file to upload example phrases for a specific action, download only a single intent. In watsonx Assistant, only one intent can be uploaded per action.

3. Specify the name and location in which to store the CSV file that is generated, and click **Save**.

You can now perform the following tasks to migrate information to watsonx Assistant:

- Upload intents as actions. For more information, see Uploading intents as actions.
- Upload the examples from a single intent as example phrases for a specific action. For more information, see Uploading phrases.

Downloading entities

You can download a number of entities to a CSV file, so you can then upload and reuse them as saved customer responses in watsonx Assistant.

1. Go to the **Entities** page.
2. Select the entities that you want to download, and then click the **Download** icon .
3. Specify the name and location in which to store the CSV file that is generated, and click **Save**.

You can now upload the entities as saved customer responses in watsonx Assistant. For more information, see Uploading saved customer responses.

Calling actions from a dialog

In watsonx Assistant, you can use actions with your primary dialog conversation. A dialog feature takes precedence over actions. You can use actions to supplement a dialog-based conversation, but the dialog drives the conversation with users to match their requests.

A dialog node calls an action to perform a task and then return to the dialog. From a single dialog node, you can make a call to either a webhook or an action, not both. Choose the approach that best suits your needs:

- Call an action if you want to perform a discrete and self-contained task, and then return to the dialog to be ready to address any new user requests. Data collected from the user during action processing must be stored as a session variable for it to be passed to the dialog.
- Call a webhook if you want to send a request to a web service to complete a task and return a response that can be used later by this dialog. For more information, see Extending your assistant with webhooks.

When to call an action from a dialog

You might want to call an action from a dialog in the following scenarios:

- You want to perform an action from multiple dialog threads.

  For example, you want to ask customers how satisfied they are with your service. You can define a single action to check customer satisfaction and call it from multiple branch-ending dialog nodes.

  In this scenario, you don't need to define an intent, such as `#check_satisfaction`. The action is called automatically, replacing a jump to another dialog node's response.

- You want to see how actions works.

  In this scenario, you can pick an intent for the action to handle. If you plan for the action to be called from the dialog only, you can spend your time defining the user examples for the intent that triggers the dialog node. When you define the action, you can add one customer example only, reducing the time you need to spend building the action.

Adding a call to an action

Before you start, open the **Actions** page and check the following:

- The name of the action you want to call.
- If the action you want to call uses a session variable, and you want to pass a value to the action by setting the session variable value, make a note of the session variable's ID.

  In the Variables section, click to open the session variable. You can see the syntax that is used for the variable name in the **ID** field. Copy the variable ID to the clipboard.

  You also need to understand the type of value to assign to the session variable. If you don't know, check how the session variable is used by the action.

To call an action from a dialog node:

1. From the dialog node, click **Customize**.
2. Set **Call out to webhooks / actions** to **On**.
3. Select **Call an action**.

   When you add a call to an action, multiple conditional responses are enabled for the dialog node automatically.
4. Click **Apply**.
5. In the dialog node, choose the action you want to call.
6. **Optional**: If the action you want to call uses a session variable, you can set the value of the session variable when the action is started.

   In the **Parameters** section, add a key and value pair to pass with the call to the action.

   Add the variable ID (that you copied to the clipboard earlier) to the **Key** field, and in the **Value** field, specify the value that you want to set the variable to.

   For example, let's say the action has a session variable named `given name`. You can pass the current user's given name (which you asked for and saved to a `$name` context variable) to the action as follows:

Key	Value
`given_name`	`"$name"`

Action call key and value pair example

7. If you call actions from more than one dialog node, change the name of the return variable to make it unique across all of your dialog nodes. For

example, you might already call an action and its return variable is called `$action_result_1`, so you can name the new one `$action_result_2`.

The return variable is a context variable that is automatically created. It stores any session variable values that are created or have their values changed while the called action is processed.

8. In the **Assistant responds** section, the condition for the first conditional response is populated automatically with the return variable from the action call.

   The return variable is named `$action_result_1` unless you change the name.

   If any session variables are created or have their values changed by the called action, then they are returned and stored in the result variable as a JSON object.

   For example, if the action changes the value of the `given name` and `membership status` session variables, the following JSON object is returned:

   ```
   {"given_name":"Sally","membership_status":"true"}
   ```

9. If you don't send or change any session variables, you can delete the first conditional response. Otherwise, consider adding a response to show when a return variable is sent from the called action.

   You might want to specify a custom response to show in case the exchange that took place when the action was processed resulted in a change to the value of the session variable that you passed to the action.

   For example, let's say that the dialog asks for the person's given name and stores it in the `$name` context variable. The dialog then passes the name to the action as a `given_name` session variable. Then, the action that is called asks if the customer prefers that the assistant use a nickname. The action then replaces the value that was stored in the `given_name` session variable with the nickname that the customer submitted.

   To continue with this example, you might want to address the user by the preferred nickname, now that you know it. You can add a response that says, `Thanks for your business, <? $action_result_1.given_name ?>!` The `<? $action_result_1.given_name ?>` expression extracts the value of the `given_name` session variable that is returned from the called action.

10. Add a response to show when no return variable is provided as the second conditional response.

    The second response that is added automatically for you has an `anything_else` condition. This response is shown if none of the other conditional responses are displayed.

    You can delete or edit the conditional responses that are added automatically. You can add more conditional responses and reorder them.

11. Click **X** to close the dialog node. Your changes are saved automatically.

12. Use the **Preview** page to test the interaction between the dialog and the action.

Analyzing dialog and actions

The **Analyze** page provides a summary of the interactions between users and your assistant. If the dialog feature is enabled, the **Analyze** page remains the same, but some slight differences in functionality exist.

Note: You can search and analyze the conversational search requests from the customer in the **Analyze** page.

Overview tab

When you view the **Overview** page, you can see action completion information in the **Action completion** diagram if a dialog node triggers an action. The **Action completion** diagram is empty if you are using only a dialog in your assistant. The three cards that display information about the most frequent actions, least frequent actions, and least completed actions are not available if your assistant uses only a dialog.

For more information about the **Analyze** page and how to use analytics with actions, see Use analytics to review your entire assistant at a glance.

Action completion tab

The **Action completion** page of watsonx Assistant provides an overview of how all your assistant's actions are doing. If the dialog feature is enabled, the **Action completion** tab is relevant only if a dialog node triggers an action. If your assistant uses only a dialog, then this tab will be empty.

For more information about understanding action completion with actions, see Understand your most and least successful actions.

Conversations tab

The **Conversations** page of watsonx Assistant provides a history of conversations between users and a deployed assistant. You can use this history to

improve how your assistants understand and respond to user requests.

From the **Conversations** page, an intent that directly calls an action is displayed in the **Topics** column. For example, you might set up an intent called `#buy_takeout` in the dialog, and that intent calls the `order pizza` action. This conversation topic is listed as `#buy_takeout > order pizza` in the **Topics** column.

You might also see **Dialog called action** listed in the **Requests** column next to a conversation. In this case, customer input triggered an intent. Then, the customer engaged with the assistant before an action was eventually called.

For more information about analyzing conversations with actions, see Review customer conversations.

Publishing dialog and actions

If the dialog feature is enabled, the publishing and deployment processes remain the same. However, some slight differences in functionality exist.

To learn about the overall publishing and deployment model for watsonx Assistant, see the Publishing and deploying your assistant overview. For more information about the **Publish** page and how the publishing process works, see Publishing your content. The following slight differences exist when the dialog feature is enabled in an assistant:

- On the **Publish** page, the information in the **Content type** column lists whether your content changes contain a dialog.
- The version tiles on the **Publish** and **Environments** pages show whether the published content contains actions, or actions and a dialog. For example, if the dialog feature is enabled in your assistant, the version tile displays `Contains actions & dialog`.
- When you export a version of your content from the **Publish** page, two JSON files are downloaded. One file is for actions and one file is for the dialog.

Building your assistant with dialog

Adding dialog

You can add one dialog skill to an assistant.

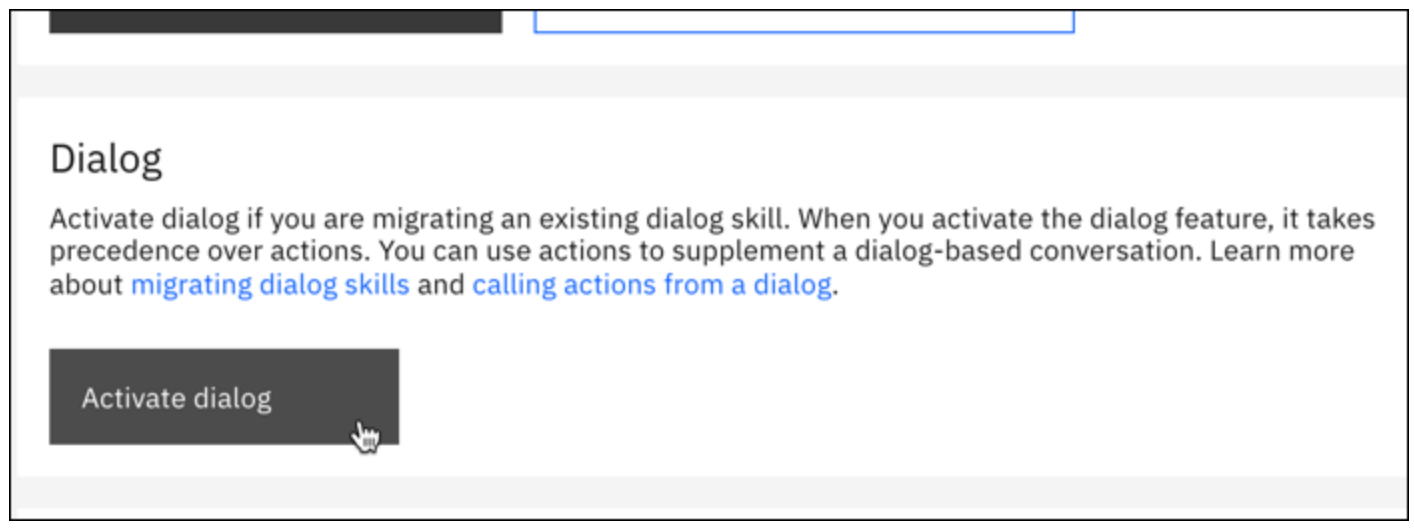
- If you are using watsonx Assistant, you need to activate dialog.
- If you are using the classic experience, you create a dialog skill.

Activating dialog

Note: If you need to support both the Dialog skill and the Action skill in the same assistant, use the `Activate dialog` button. This button enables a phased migration strategy for gradually transitioning an assistant built on dialog to Actions.

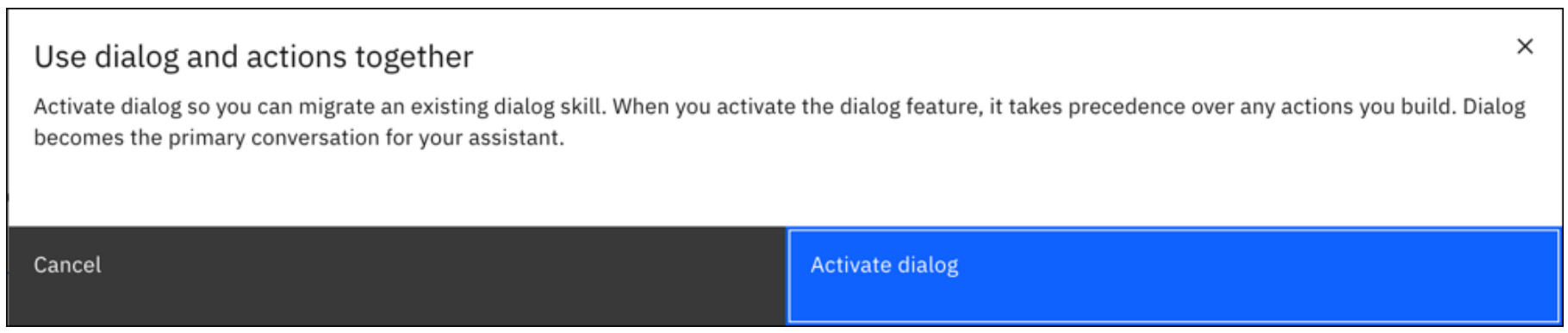
To activate the dialog feature in watsonx Assistant:

1. Create or open an assistant where you want to use a dialog as the primary conversation with users.
2. Open **Assistant settings** ⚙.
3. Click **Activate dialog**.



Activate dialog

4. In the confirmation that displays, click **Activate dialog** again.



Activate dialog

Once you activate the dialog feature, it takes precedence over actions. You can use actions to supplement a dialog-based conversation, but the dialog drives the conversation with users to match their requests. For more information, see Calling actions from a dialog.

Create the dialog skill

To add a skill in the classic experience, complete the following steps:

1. On the Skills page, click **Create skill**.
2. Choose **Dialog skill**.
3. Do one of the following:
    - To create a new dialog skill, remain on the *Create skill* tab.
    - To add the dialog sample skill that is provided with the product as a starting point for your own skill or as an example to explore before you create one yourself, open the *Use sample skill* tab, and then click the sample labeled **TYPE: Dialog**. You can skip the remaining steps.
    - To add a skill that was downloaded previously, you can upload it as a JSON file. Open the *Upload skill* tab. Drag a file or click **Drag and drop file**

here or click to select a file and select the JSON file you want to upload.

> ⚠ **Important:** The uploaded JSON file must use UTF-8 encoding, without byte order mark (BOM) encoding. The JSON cannot contain tabs, newlines, or carriage returns.

> **Tip:** The maximum size for a skill JSON file is 15 MB. If you need to upload a larger skill, consider using the REST API. For more information, see the API Reference.

Click **Upload**.

4. Specify the details for the skill:
   - **Name**: A name no more than 64 characters in length. A name is required.
   - **Description**: An optional description no more than 128 characters in length.
   - **Language**: The language of the user input the skill will be trained to understand. The default value is English.
5. For **Skill type**, choose Dialog.
6. Click **Create skill**.

After you create the dialog skill, it appears as a tile on the Skills page. Now, you can start identifying the user goals that you want the dialog skill to address.

Adding the skill to an assistant

You can add one dialog skill to an assistant. You must open the assistant tile and add the skill to the assistant from the assistant configuration page; you cannot choose the assistant that will use the skill from within the skill configuration page. One dialog skill can be used by more than one assistant.

1. From the Assistants page, click to open the tile for the assistant to which you want to add the skill.
2. Click **Add an actions or dialog skill** .
3. Click **Add existing skill**.

   Click the skill that you want to add from the available skills that are displayed.

When you add a dialog skill from here, you get the development version. If you want to add a specific skill version, add it from the skill's *Versions* page instead.

Actions skill in the classic experience

If you're using the classic experience, you can add an actions skill instead of a dialog skill. Actions represent the tasks you want your assistant to help your customers with.

Each action contains a series of steps that represent individual exchanges with a customer. Building the conversation that your assistant has with your customers is fundamentally about deciding which steps, or which user interactions, are required to complete an action. After you identify the list of steps, you can then focus on writing engaging content to turn each interaction into a positive experience for your customer.

If you're interested in using actions, watsonx Assistant is recommended, rather than using an actions skill in the classic experience. IBM® watsonx™ Assistant provides a simplified user interface, an improved deployment process, and access to the latest features. For more information, see Overview: Editing actions.

Search skill in the classic experience

Plus

When watsonx Assistant doesn't have an explicit solution to a problem, it routes the user question to a search skill to find an answer from across your disparate sources of self-service content. The search skill interacts with the IBM Watson® Discovery service to extract this information from a configured data collection.

If you already use the Discovery service, you can mine your existing data collections for source material that you can share with customers to address their questions.

For more information, see IBM Watson® Discovery search integration setup.

Skill limits

The number of skills you can create depends on your watsonx Assistant plan type. Any sample dialog skills that are available for you to use do not count toward your limit unless you use them. A skill version does not count as a skill.

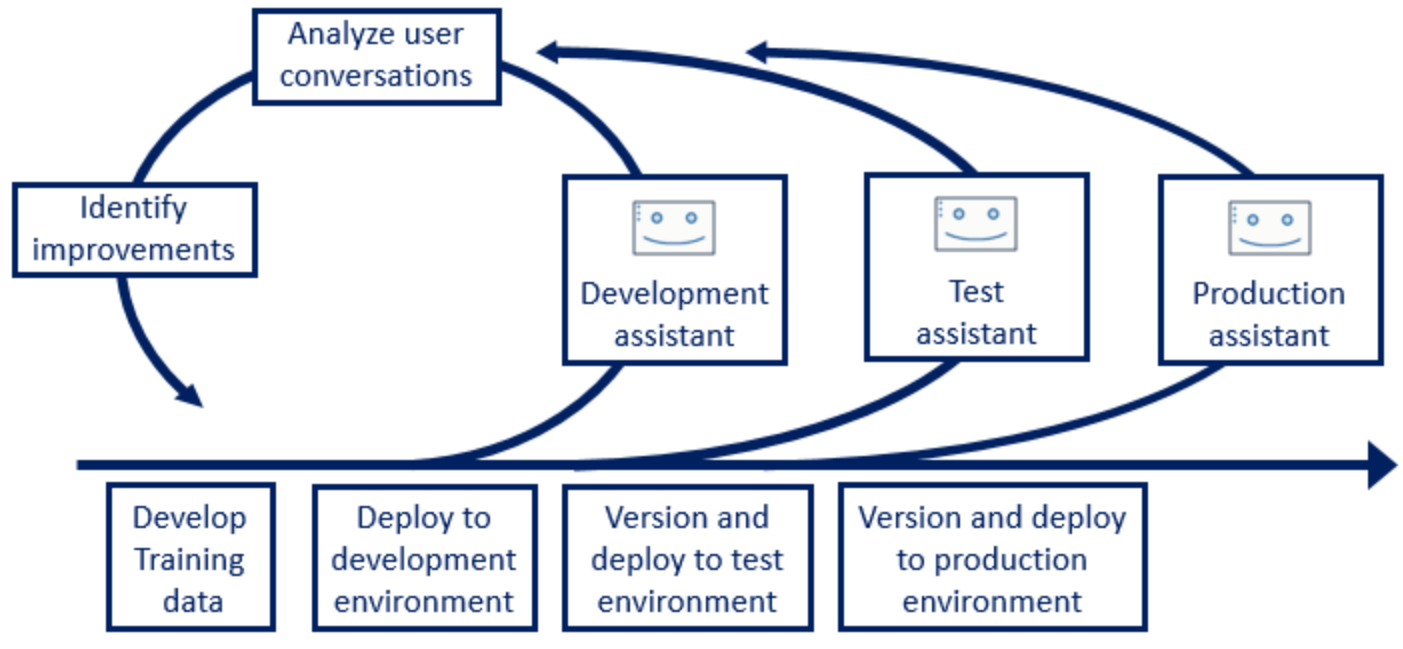
Plan	Maximum number of skills of each type per service instance
Enterprise	100
Premium (legacy)	100
Plus	50
Trial	50
Standard (legacy)	20
Lite	5

Plan details

Note: After 30 days of inactivity, an unused skill in a Lite plan service instance might be deleted to free up space.

Dialog creation workflow

Use watsonx Assistant to leverage AI as you build, deploy, and incrementally improve a conversational assistant.



Development steps

Development process

The typical workflow for an assistant project includes the following steps:

1. Define a narrow set of key customer needs that you want the assistant to address on your behalf, including any business processes that it can initiate or complete for your customers. Start small.
2. Create intents that represent the customer needs you identified in the previous step. For example, intents such as `#about_company` or `#place_order`.
3. Build a dialog that detects the defined intents and addresses them, either with simple responses or with a dialog flow that collects more information first.
4. Define any entities that are needed to more clearly understand the user's meaning. For example, you might add an `@product` entity that you can use with the `#place_order` intent to understand what product the customer wants to buy.

   Mine existing intent user examples for common entity value mentions. Using annotations to define entities captures not only the text of the entity value, but the context in which the entity value is typically used in a sentence.
5. Test each function that you add to the assistant in the "Try it" pane, incrementally, as you go.
6. When you have a working assistant that can successfully handle key tasks, add an integration that deploys the assistant to a development environment. Test the deployed assistant and make refinements.

7. After you build an effective assistant, take a snapshot of the dialog skill and save it as a version.

   Saving a version when you reach a development milestone gives you something you can go back to if subsequent changes you make to the skill decrease its effectiveness.

8. Deploy the version of the assistant into a test environment, and test it.

   If you use the preview, you can share the URL with others to get their help with testing.

9. Use metrics from the Analytics tab to find areas for improvement, and make adjustments.

   If you need to test alternative approaches to addressing an issue, create a version for each solution, so you can deploy and test each one independently, and compare the results.

10. When you are happy with the performance of your assistant, deploy the best version of the assistant into a production environment.

11. Monitor the logs from conversations that users have with the deployed assistant.

    You can view the logs for a version of a skill that is running in production from the Analytics tab of a development version of the skill. As you find misclassifications or other issues, you can correct them in the development version of the skill, and then deploy the improved version to production after testing.

The process of analyzing the logs and improving the dialog skill is ongoing. Over time, you might want to expand the tasks that the assistant can handle for you. Customer needs also change. As new needs arise, the metrics generated by your deployed assistants can help you identify and address them in subsequent iterations.

Defining goals for your assistant to address

Creating intents

Intents are purposes or goals that are expressed in a customer's input, such as answering a question or processing a bill payment. By recognizing the intent expressed in a customer's input, the watsonx Assistant service can choose the correct dialog flow for responding to it.

Intent creation overview

- Plan the intents for your application.

  Consider what your customers might want to do, and what you want your application to be able to handle on their behalf. For example, you might want your application to help your customers make a purchase. If so, you can add a `#buy_something` intent. (The `#` that is added as a prefix to the intent name helps to clearly identify it as an intent.)

- Teach watsonx Assistant about your intents.

  After you decide which business requests that you want your application to handle for your customers, you must teach watsonx Assistant about them. For each business goal (such as `#buy_something`), you must provide at least 5 examples of utterances that your customers typically use to indicate their goal. For example, `I want to make a purchase.`

  Ideally, find real-world user utterance examples that you can extract from existing business processes. Tailor the user examples to your specific business. For example, if you are an insurance company, a user example might look more like this, `I want to buy a new XYZ insurance plan.`

  Your assistant uses the examples that you provide to build a machine learning model that can recognize the same and similar types of utterances and map them to the appropriate intent.

Start with a few intents, and test them as you iteratively expand the scope of the application.

Creating intents

1. Click **Intents**.
2. Select **Create intent**.
3. In the **Intent name** field, type a name for the intent.
   - The intent name can contain letters (in Unicode), numbers, underscores, hyphens, and periods.
   - The name cannot consist of `..` or any other string of only periods.
   - Intent names cannot contain spaces and must not exceed 128 characters. The following are examples of intent names:
     - `#weather_conditions`
     - `#pay_bill`

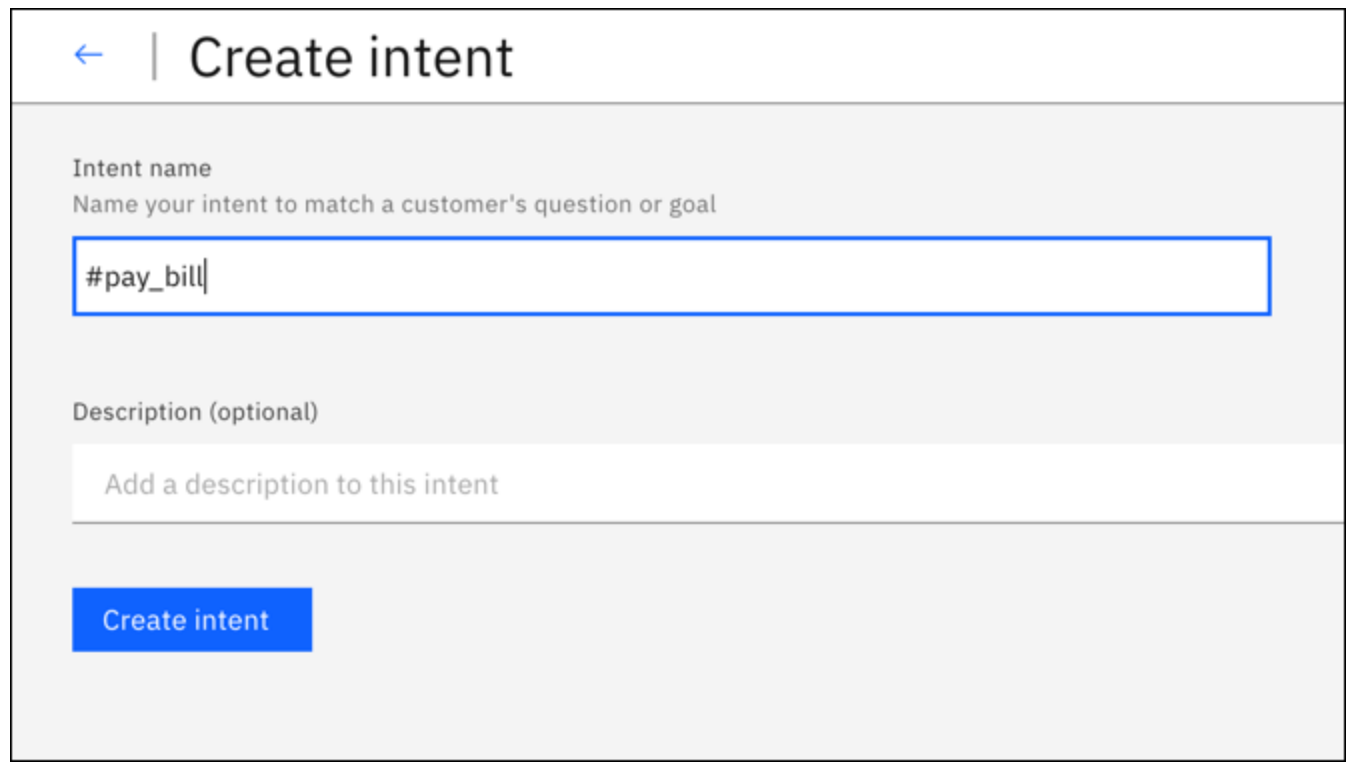
- `#escalate_to_agent`

> **Tip:** A number sign `#` prefix is included in the intent name automatically to help identify the term as an intent. You do not need to add it.

Keep the name as short as possible for readability in the "Try it out" pane and conversation logs.

Optionally add a description of the intent in the **Description** field.

4. Select **Create intent** to save your intent name.



New intent

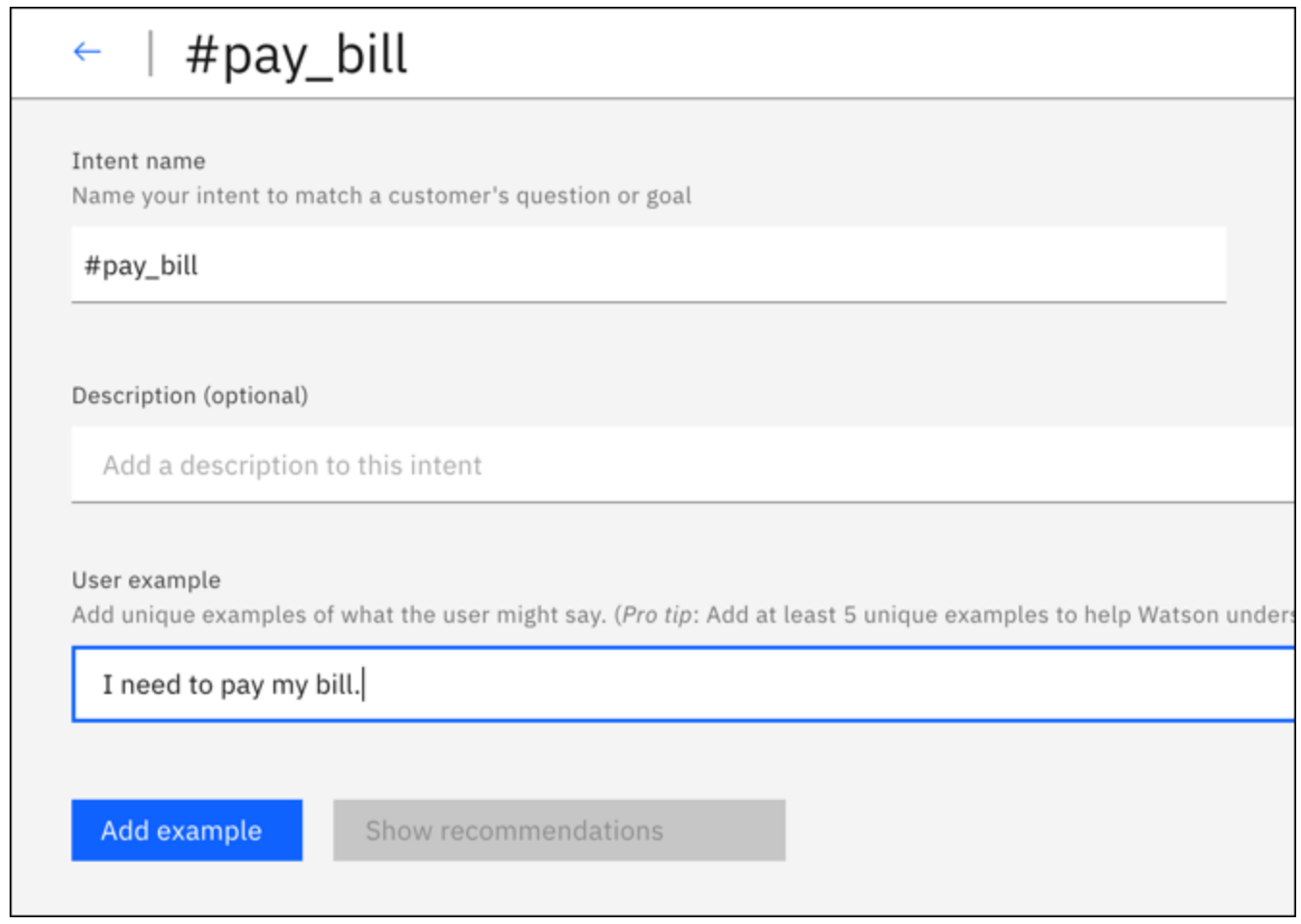
5. In the **User example** field, type the text of a user example for the intent. An example can be any string up to 1,024 characters in length. The following utterances might be examples for the `#pay_bill` intent:

   - `I need to pay my bill.`
   - `Pay my account balance`
   - `make a payment`

> **Tip:** To learn about the impact of including references to entities in your user examples, see How entity references are treated.

> **Important:** Intent names and example text can be shown in URLs when an application interacts with watsonx Assistant. Do not include sensitive or personal information in these artifacts.

6. Click **Add example** to save the user example.



Add user example

7. Repeat the same process to add more examples.

   > ⚠ **Important:** Provide at least five examples for each intent.

8. When you are done adding examples, click the close arrow ← to finish creating the intent.

The system trains itself on the intent and user examples you added.

Important:

- Intent example data should be representative and typical of data that your users provide. Examples can be collected from actual user data, or from people who are experts in your specific field. The representative and accurate nature of the data is important.
- Both training and test data (for evaluation purposes) should reflect the distribution of intents in real usage. Generally, more frequent intents have relatively more examples, and better response coverage.
- You can include punctuation in the example text if it appears naturally. If you believe that some users express their intents with examples that include punctuation, and some users will not, include both versions. Generally, the more coverage for various patterns, the better the response.

How entity references are treated

When you include an entity mention in a user example, the machine learning model uses the information in different ways in these scenarios:

- Referencing entity values and synonyms in intent examples
- Annotated mentions
- Directly referencing an entity name in an intent example

Referencing entity values and synonyms in intent examples

If you define, or plan to define, entities that are related to this intent, mention the entity values or synonyms in some of the examples. Doing so helps to establish a relationship between the intent and entities. It is a weak relationship, but it does inform the model.

Annotated mentions

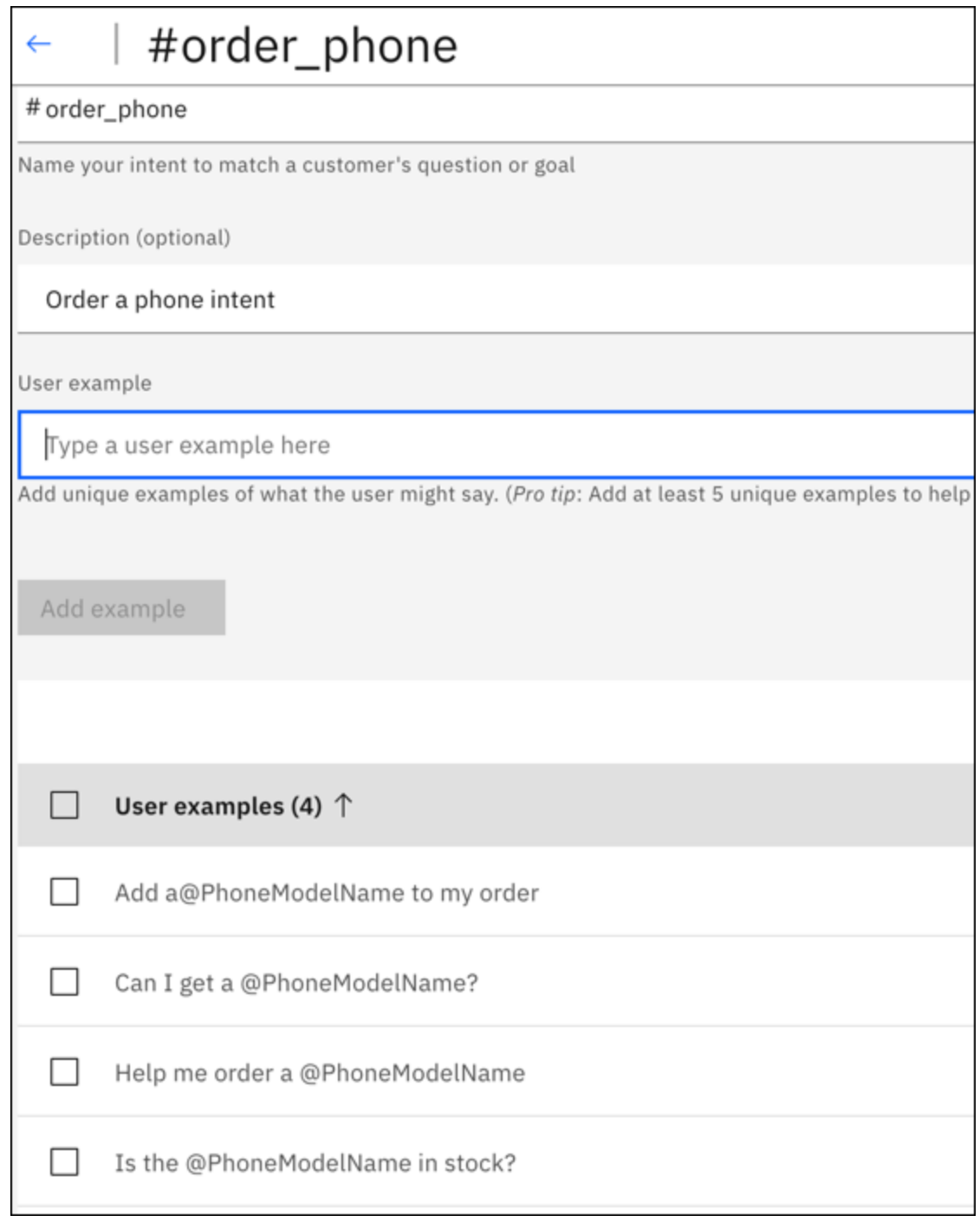
As you define entities, you can annotate mentions of the entity directly from your existing intent user examples. A relationship that you identify in this way between the intent and the entity is *not* used by the intent classification model. However, when you add the mention to the entity, it is also added to that entity as new value. And when you add the mention to an existing entity value, it is also added to that entity value as a new synonym. Intent classification does use these types of dictionary references in intent user examples to establish a weak reference between an intent and an entity.

Directly referencing an entity name in an intent example

> **Note:** This approach is advanced. If used, it must be used consistently.

You can choose to directly reference entities in your intent examples. For instance, say that you have an entity that is called `@PhoneModelName`, which contains values *Galaxy S8*, *Moto Z2*, *LG G6*, and *Google Pixel 2*. When you create an intent, for example `#order_phone`, you might then provide training data as follows:

- Can I get a `@PhoneModelName`?
- Help me order a `@PhoneModelName`.
- Is the `@PhoneModelName` in stock?
- Add a `@PhoneModelName` to my order.



Intent definition

Currently, you can only directly reference synonym entities that you define (pattern values are ignored). You cannot use system entities.

> ⚠ **Important:** If you choose to reference an entity as an intent example (for example, `@PhoneModelName`) *anywhere* in your training data it cancels the value of using a direct reference (for example, *Galaxy S8*) in an intent example anywhere else. All intents will then use the entity-as-an-intent-example approach. You cannot apply this approach for a specific intent only.

In practice, this means that if you have previously trained most of your intents based on direct references (*Galaxy S8*), and you now use entity references (`@PhoneModelName`) for just one intent, the change impacts your previous training. If you do choose to use `@Entity` references, you must replace all previous direct references with `@Entity` references.

Defining one example intent with an `@Entity` that has 10 values that are defined for it **does not** equate to specifying that example intent 10 times. The watsonx Assistant service does not give that much weight to that one example intent syntax.

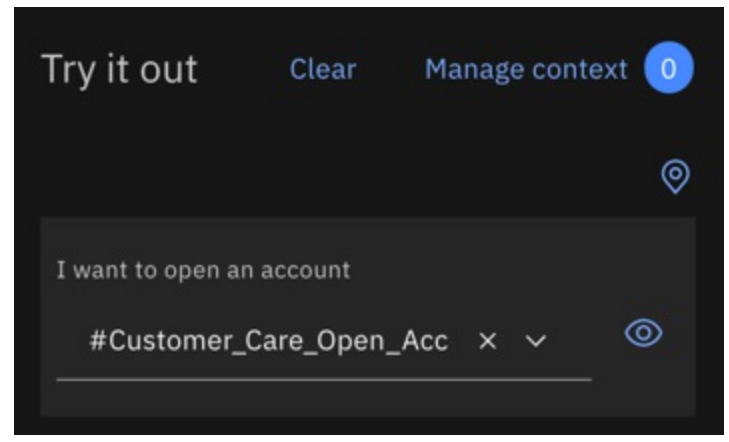
Testing your intents

After you finish creating new intents, you can test the system to see if it recognizes your intents as you expect.

1. Click **Try it**.
2. In the "Try it out" pane, enter a question or other text string and press Enter to see which intent is recognized. If the wrong intent is recognized, you can improve your model by adding this text as an example to the correct intent.

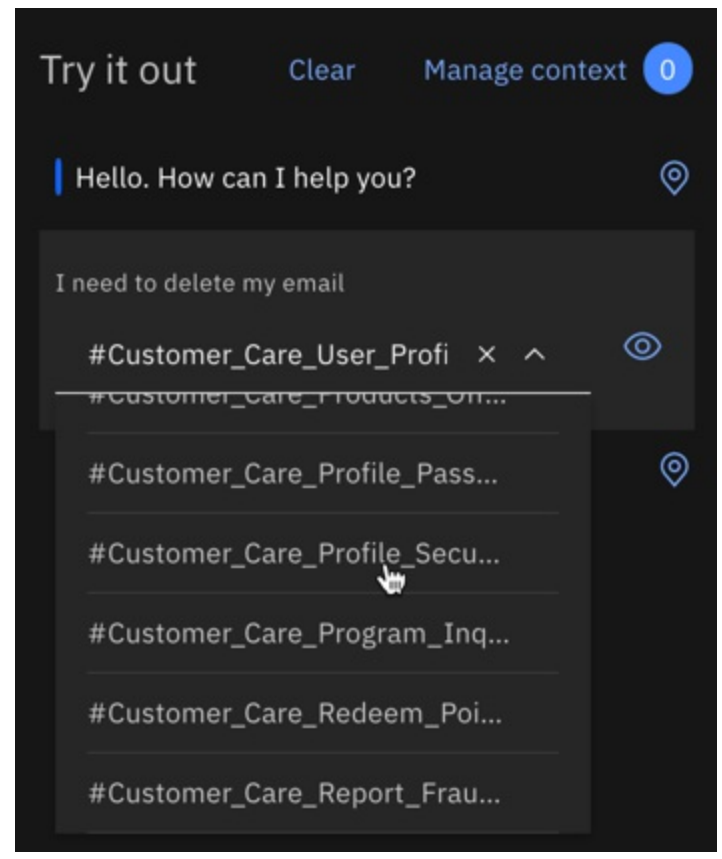
   > **Tip:** If you make recent changes in your skill, you might see a message that indicates that the system is still retraining. If you see this message, wait until training completes before testing:

   The response indicates which intent was recognized from your input.



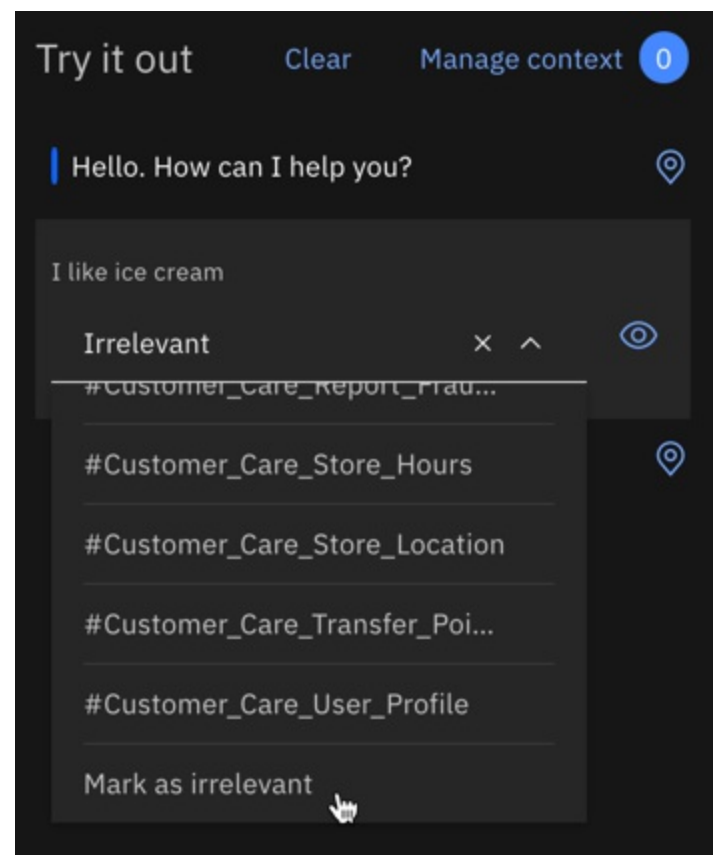
Test intents

3. If the system does not recognize the correct intent, you can correct it. To correct the recognized intent, select the displayed intent and then select the correct intent from the list. After your correction is submitted, the system automatically retrains itself to incorporate the new data.



Correcting an intent

4. If the input is unrelated to any of the intents in your application, you can teach your assistant that by selecting the displayed intent, and then clicking **Mark as irrelevant**.



Mark as irrelevnt

If your intents are not being correctly recognized, consider making the following kinds of changes:

- Add the unrecognized text as an example to the correct intent.
- Move existing examples from one intent to another.
- Consider whether your intents are too similar, and redefine them.

Absolute scoring

The watsonx Assistant service scores each intent's confidence independently, not in relation to other intents. This approach adds flexibility; multiple intents can be detected in a single user input. It also means that the system might not return an intent at all. If the top intent has a low confidence score (less than 0.2), the top intent is included in the intents array that is returned by the API, but any nodes that condition on the intent are not triggered. If you want to detect the case when no intents with good confidence scores were detected, use the `irrelevant` special condition in your dialog node.

As intent confidence scores change, your dialogs might need restructuring. For example, if a dialog node uses an intent in its condition, and the intent's confidence score starts to consistently drop below 0.2, the dialog node stops being processed. If the confidence score changes, the behavior of the dialog can also change.

Intent limits

The number of intents and examples you can create depends on your watsonx Assistant plan type:

Plan	Intents per skill	Examples per skill
Enterprise	2,000	25,000
Premium (legacy)	2,000	25,000
Plus	2,000	25,000
Trial	100	25,000
Lite	100	25,000

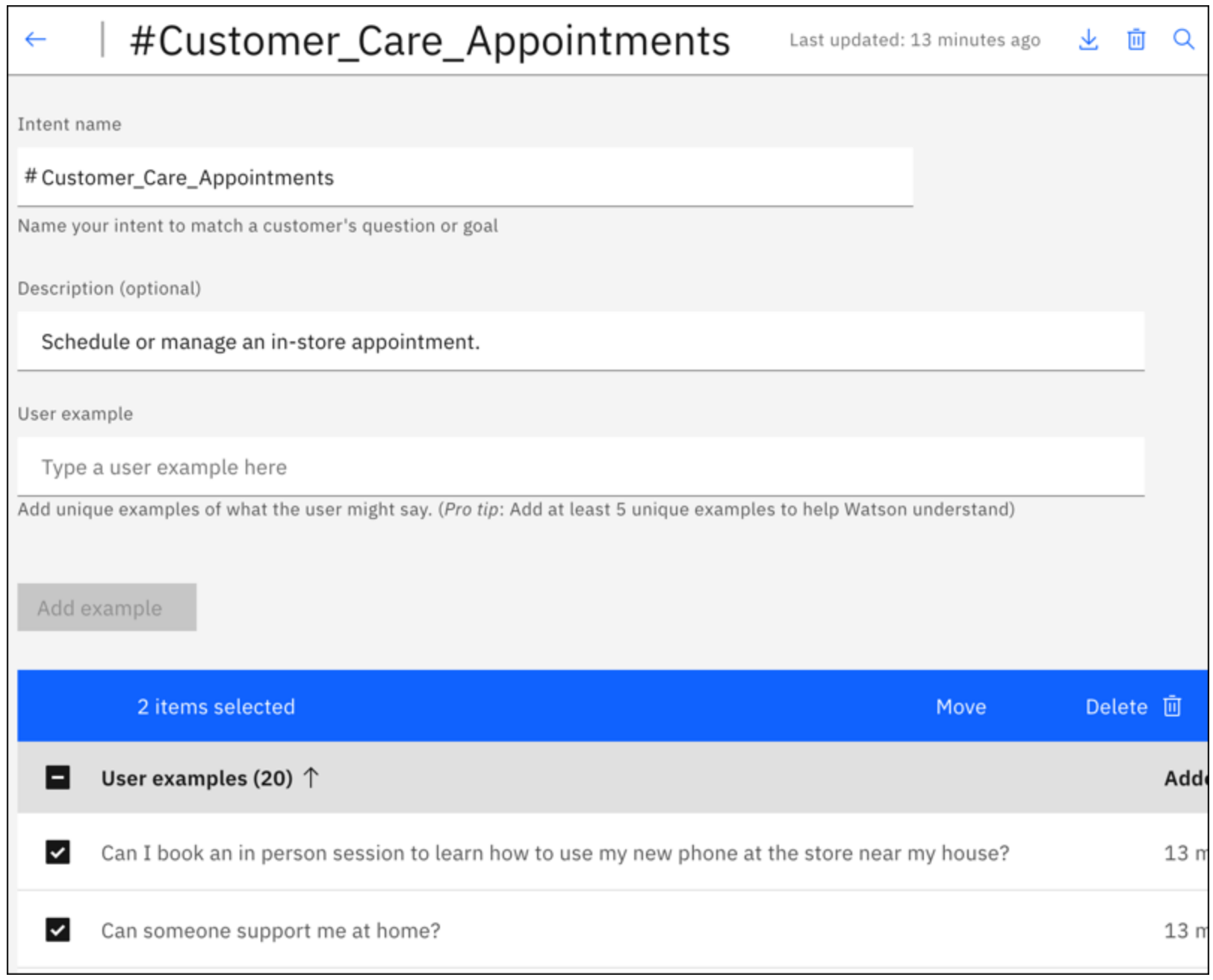
Plan details

Editing intents

You can click any intent in the list to open it for editing. You can make the following changes:

- Rename the intent
- Delete the intent
- Add, edit, or delete examples
- Move examples to a different intent

To move or delete an example, click the checkbox that is associated with it, and then click **Move** or **Delete**.



Move or delete example

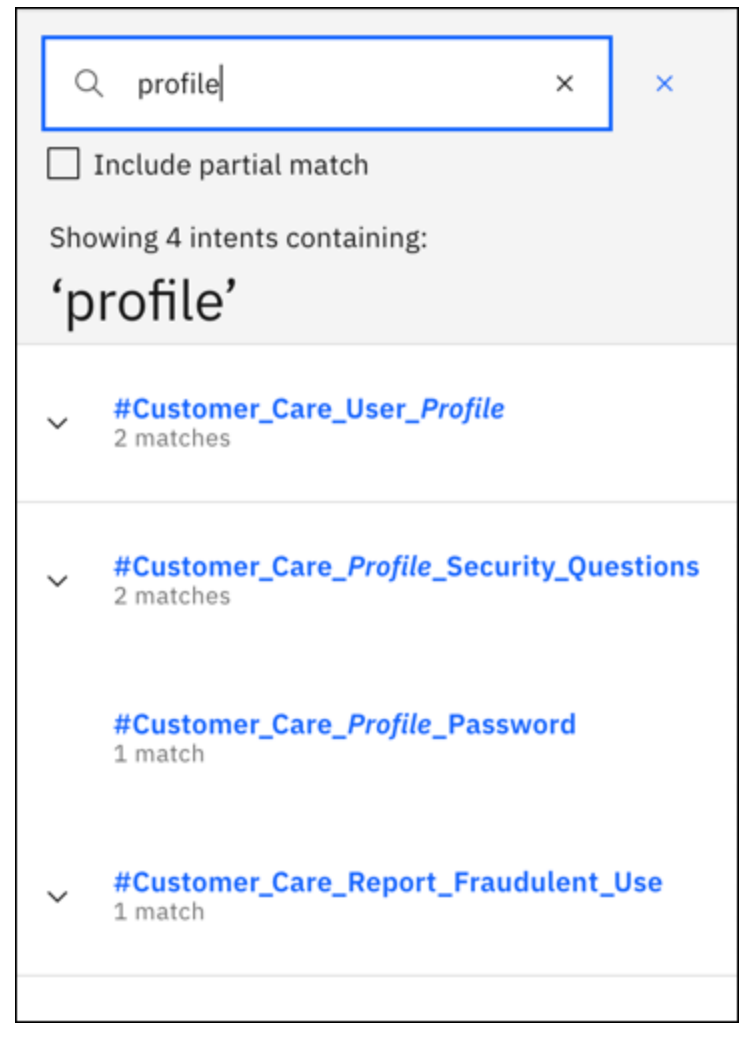
Searching intents

Use the Search feature to find user examples, intent names, and descriptions.

1. From the **Intents** page, click the Search icon .
2. Submit a search term or phrase. You can also select **Include partial match**.

   The first time that you search for something, you might get a message that says the content is being indexed. If so, wait a minute, and then resubmit the search term.

   Intents that contain your search term are displayed.



Search results

Downloading intents

You can download a number of intents to a CSV file, so you can then upload and reuse them in another watsonx Assistant application.

1. Go to the **Intents** page.

- To download all intents, meaning the intents that are listed on this and any additional pages, do not select any individual intents. Instead, click the *Download all intents* icon .
- To download the intents that are listed on the current page only, select the checkbox in the header. This action selects all of the intents on the current page. Then, click the **Download** button.
- To download one or more specific intents, select the intents that you want to download, and then click the **Download** button.

1. Specify the name and location in which to store the CSV file that is generated, and then click **Save**.

Uploading intents and examples

If you have many intents and examples, you might find it easier to upload them from a comma-separated value (CSV) file than to define them one by one. Be sure to remove any personal data from the user examples that you include in the file.

1. Collect the intents and examples into a CSV file, or export them from a spreadsheet to a CSV file. The required format for each line in the file is as follows:

   ```
   <example>,<intent>
   ```

   where `<example>` is the text of a user example, and `<intent>` is the name of the intent you want the example to match. For example:

   ```
   Tell me the current weather conditions.,weather_conditions
   Is it raining?,weather_conditions
   What's the temperature?,weather_conditions
   Where is your nearest location?,find_location
   Do you have a store in Raleigh?,find_location
   ```

   Important: Save the CSV file with UTF-8 encoding and no byte order mark (BOM).

2. From the **Intents** page, click the **Upload intents** icon .
3. Drag a file or browse to select a file from your computer.

   Important: The maximum CSV file size is 10 MB. If your CSV file is larger, consider splitting it into multiple files and uploading them separately.

4. Click **Upload intents**.

   The file is validated and uploaded, and the system trains itself on the new data.

You can view the uploaded intents and the corresponding examples on the **Intents** tab. You might need to refresh the page to see the new intents and examples.

Deleting intents

You can select a number of intents for deletion.

> ⚠ **Important:** By deleting intents that you are also deleting all associated examples, and these items cannot be retrieved later. All dialog nodes that reference these intents must be updated manually to no longer reference the deleted content.

1. Go to the **Intents** page

   - To delete all intents, meaning the intents that are listed on this and any additional pages, do not select any individual intents. Instead, click the *Delete all intents* icon.
   - To delete the intents that are listed on the current page only, select the checkbox in the header. This action selects all of the intents that are listed on the current page. Click **Delete**.
   - To delete one or more specific intents, select the intents that you want to delete, and then click **Delete**.

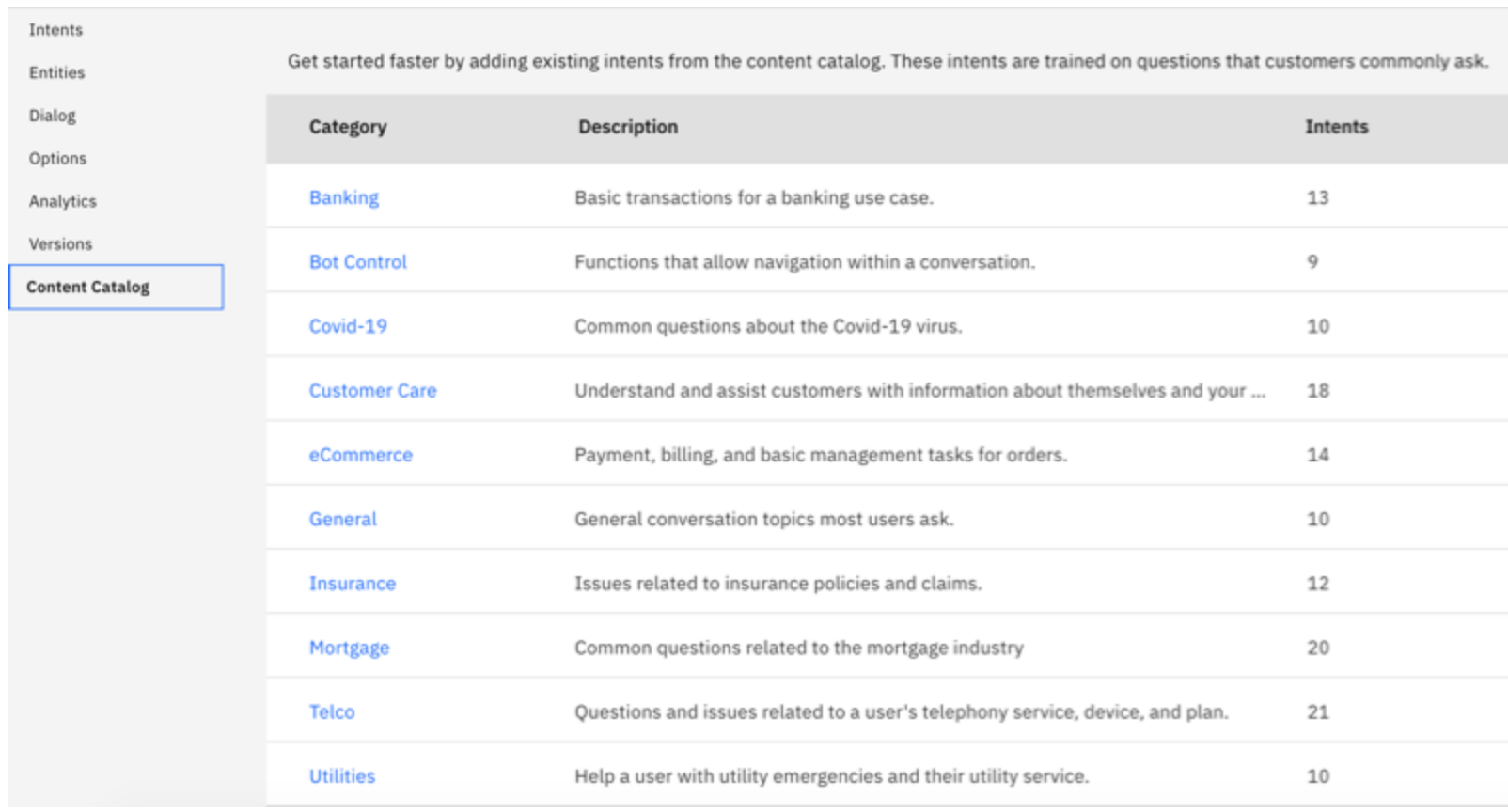
Using built-in intents

Content catalogs provide a way to add common intents to your watsonx Assistant dialog skill.

Intents that you add from the catalog are meant to provide a starting point. Add to or edit the catalog intents to tailor them for your use case.

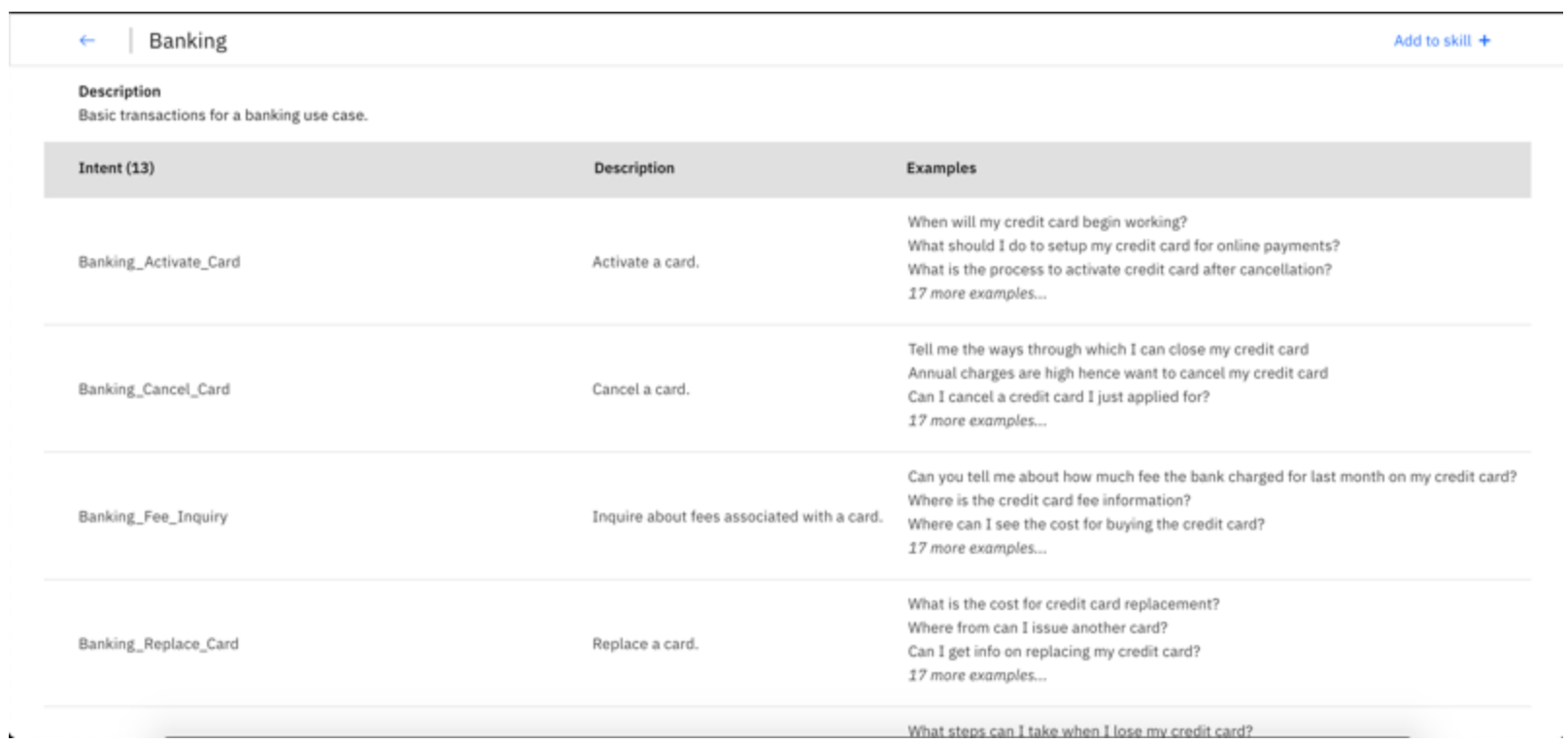
Adding a content catalog to your dialog skill

1. Open the **Content Catalog** page.



Content catalogs

2. Select a content catalog, such as *Banking*, to see the intents that are provided with it.



Intents

   Intents that are added from a content catalog are distinguishable from other intents by their names. Each intent name has a prefix with the content catalog name.

3. Add a content catalog by clicking **Add content +**.
4. Go to the **Intents** page to see the intents that you added from the catalog listed.

> **Note:** After you add a catalog, the intents become part of your training data. If IBM makes subsequent updates to a content catalog, the changes are not automatically applied to any intents you added from a catalog.

Editing content catalog intents

Like any other intent, after you add content catalog intents to your skill, you can make the following changes to them:

- Rename intents.
- Delete intents.
- Add, edit, or delete intent user examples.

- Move an example to a different intent.

Building a conversational flow

How your dialog is processed

The dialog uses the intents that are identified in the user's input, plus context from the application, to interact with the user and ultimately provide a useful response.

The dialog matches intents (what users say they want to do) to responses (what the bot says back). The response might be the answer to a question such as `What are your store hours?` or the execution of a command, such as placing an order. The intent and entity might be enough information to identify the correct response, or the dialog might ask the user for more input that is needed to respond correctly. For example, if a user asks, `Where can I get some food?` you might want to clarify whether they want a restaurant or a grocery store, to dine in or take out, and so on. You can ask for more details in a text response and create one or more child nodes to process the new input.

The dialog is represented graphically in watsonx Assistant as a tree. Create a branch to process each intent that you want your conversation to handle. A branch is composed of multiple nodes.

Dialog nodes

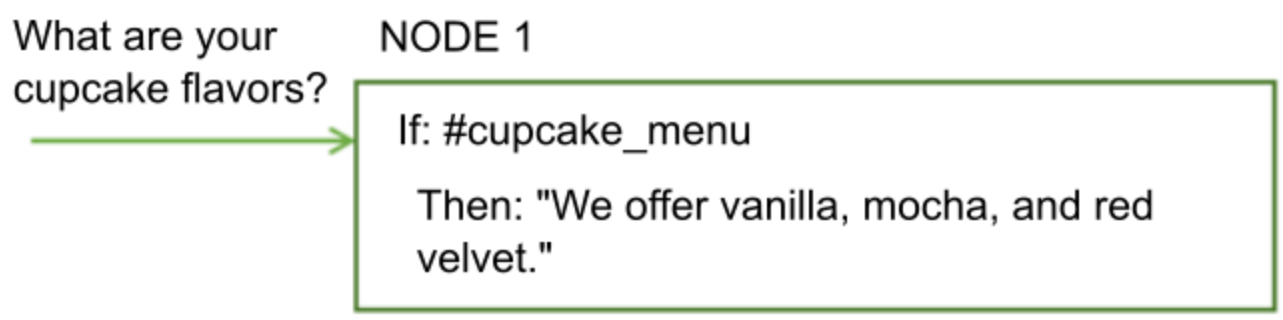
Each dialog node contains, at a minimum, a condition and a response.

![Shows user input going to a box that contains the statement If: CONDITION, Then: RESPONSE](images/node1-empty.png

- Condition: Specifies the information that must be present in the user input for this node in the dialog to be triggered. The information is typically a specific intent. It might also be an entity type, an entity value, or a context variable value. For more information, see Conditions.
- Response: The utterance that your assistant uses to respond to the user. The response can also be configured to show an image or a list of options, or to trigger programmatic actions. For more information, see Responses.

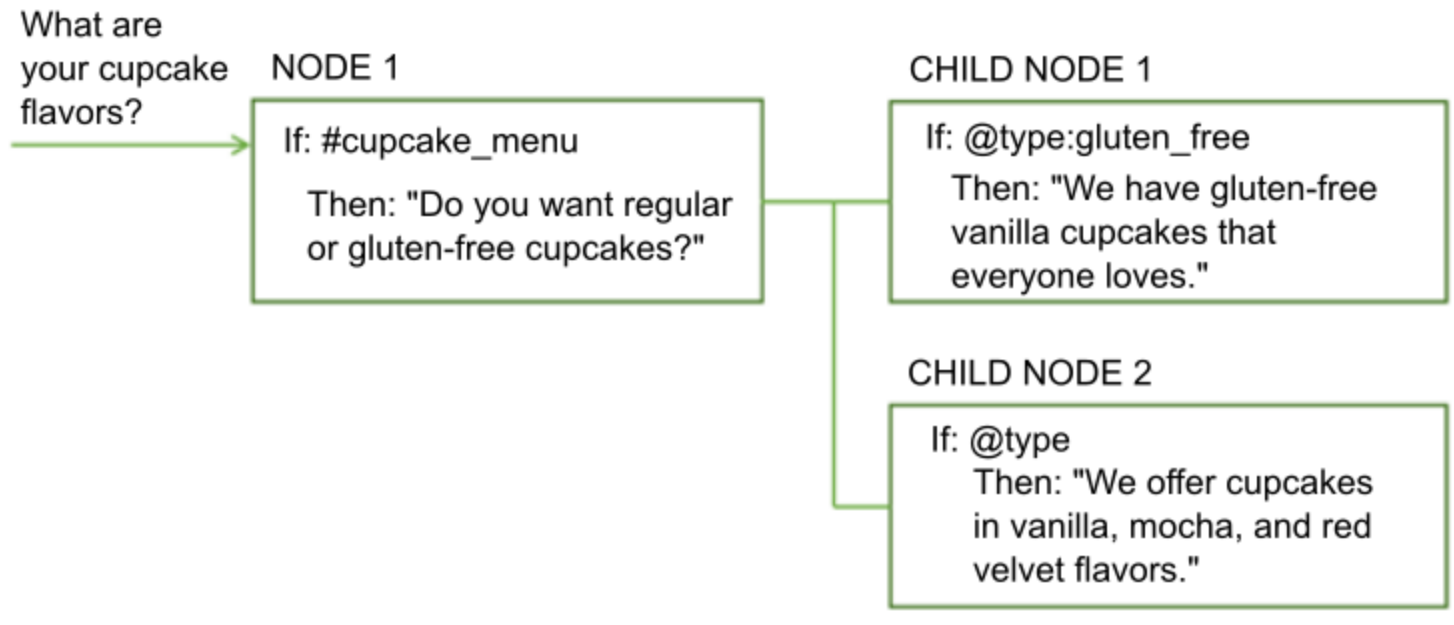
You can think of the node as having an if/then construction: if this condition is true, then return this response.

For example, the following node is triggered if the natural language processing function of your assistant determines that the user input contains the `#cupcake-menu` intent. As a result of the node being triggered, your assistant responds with an appropriate answer.



Simple condition and response

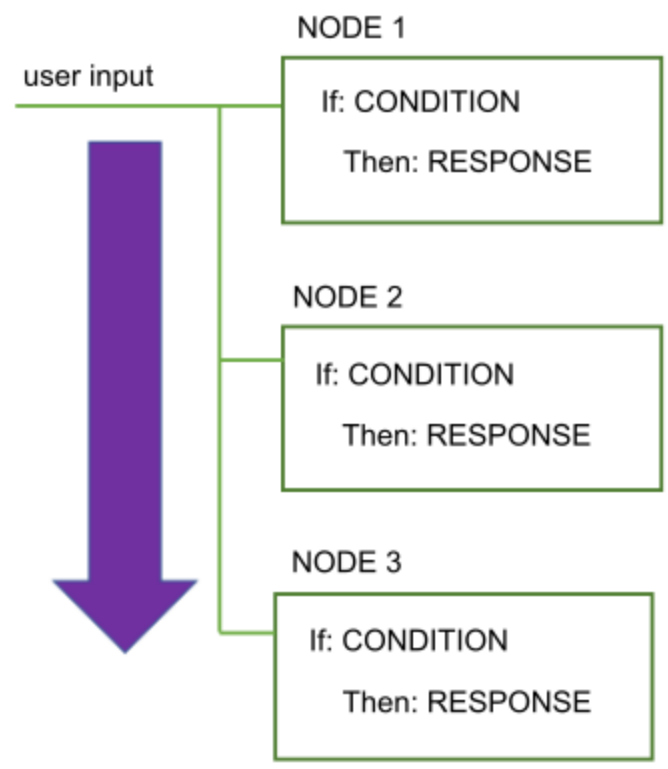
A single node with one condition and response can handle simple user requests. But, more often than not, users have more sophisticated questions or want help with more complex tasks. You can add child nodes that ask the user to provide any additional information that your assistant needs.



Node with child nodes

Dialog flow

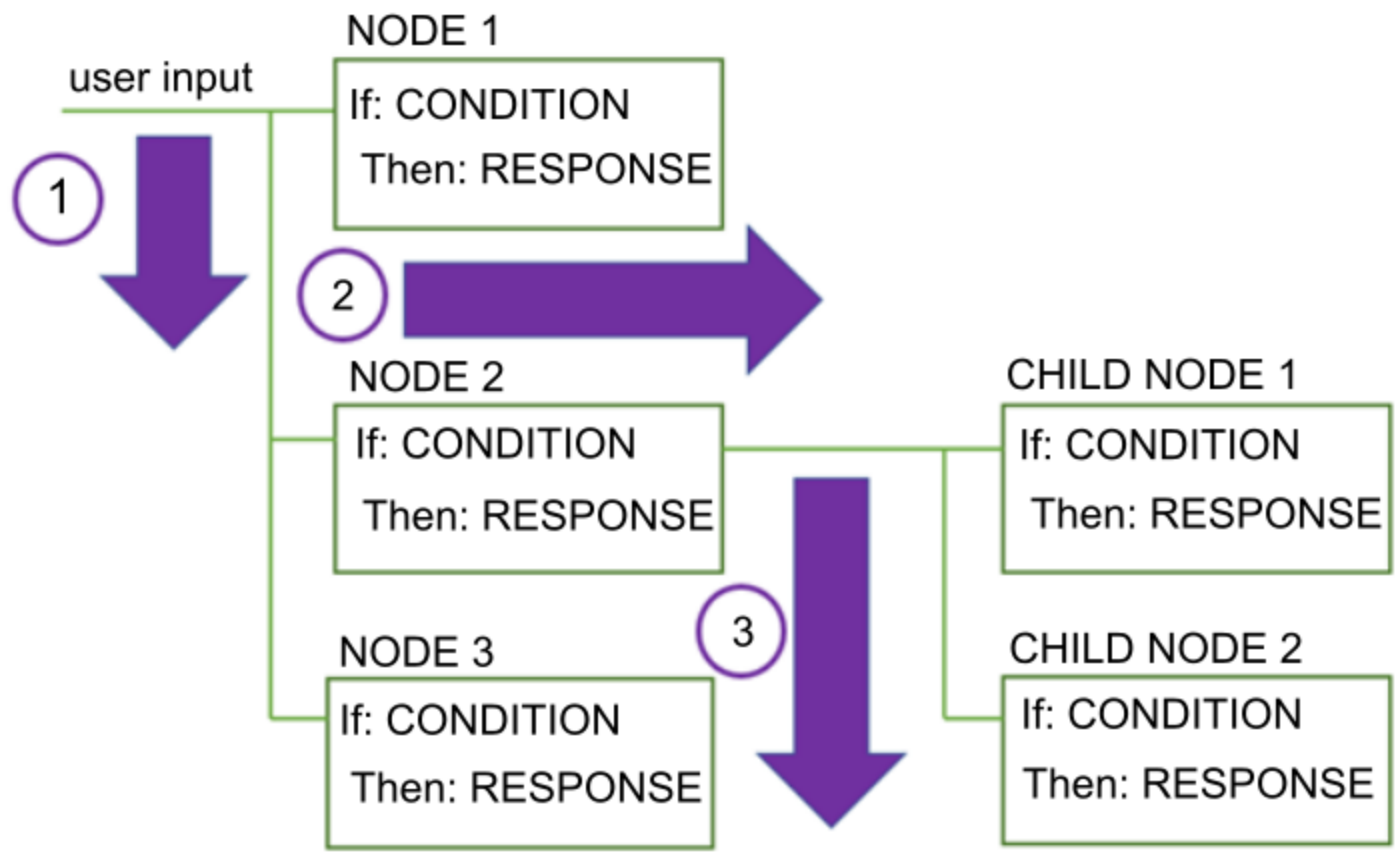
The dialog that you create is processed by your assistant from the first node in the tree to the last.



Node flow

As it travels down the tree, if your assistant finds a condition that is met, it triggers that node. It then moves along the triggered node to check the user input against any child node conditions. As it checks the child nodes, it moves again from the first child node to the last.

Your assistant continues to work its way through the dialog tree from first to last node, along each triggered node, then from first to last child node, and along each triggered child node until it reaches the last node in the branch it is following.



Node flow

When you start to build the dialog, you must determine the branches to include, and where to place them. The order of the branches is important because nodes are evaluated from first to last. The first root node whose condition matches the input is used; any nodes that come later in the tree are not triggered.

When your assistant reaches the end of a branch, or cannot find a condition that evaluates to true from the current set of child nodes it is evaluating, it jumps back out to the base of the tree. Your assistant processes the root nodes from first to the last. If none of the conditions evaluates to true, then the response from the last node in the tree, which typically has a special `anything_else` condition that always evaluates to true, is returned.

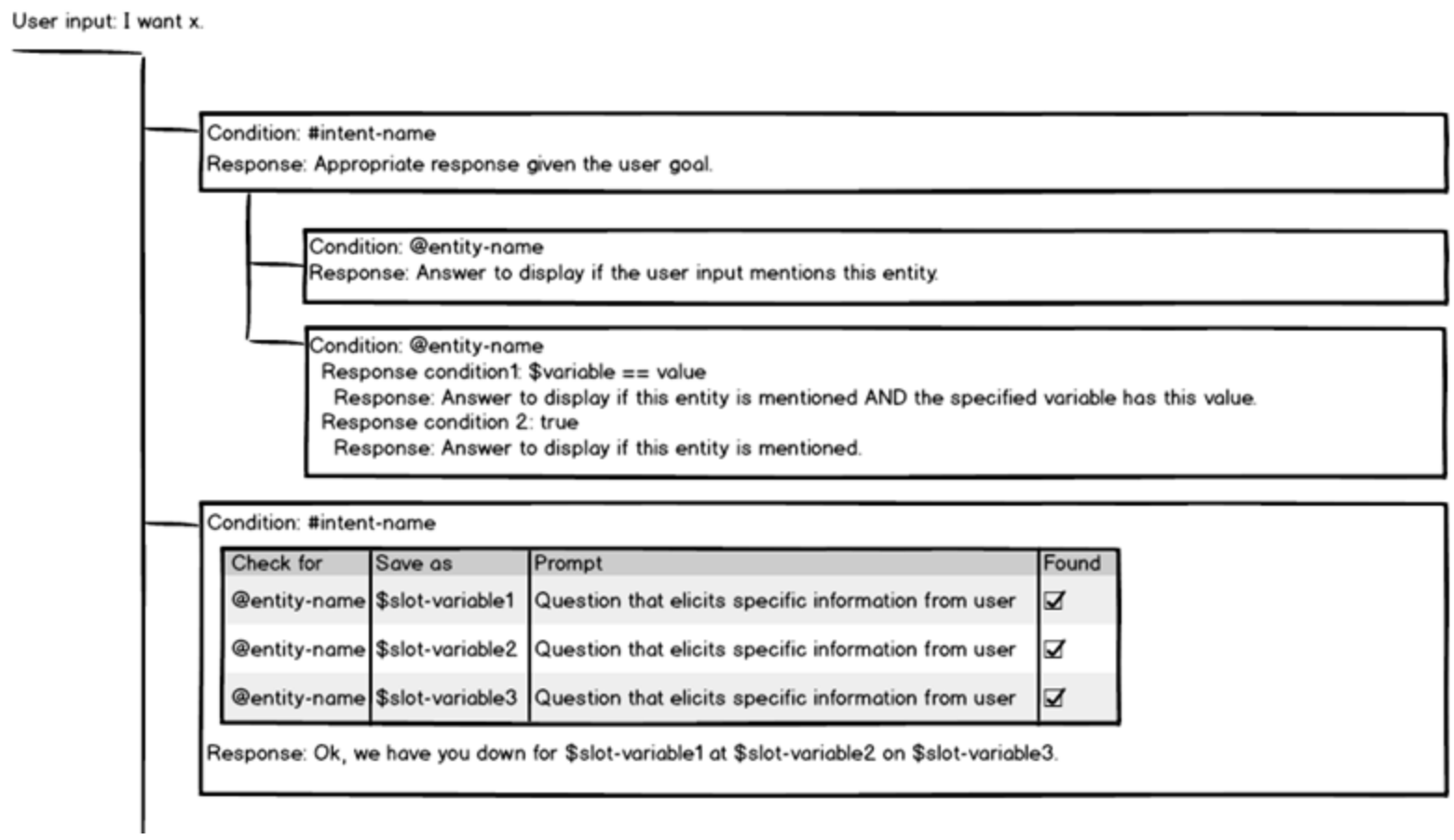
You can disrupt the standard first-to-last flow in the following ways:

- By customizing what happens after a node is processed. For example, you can configure a node to jump directly to another node after it is processed, even if the other node is positioned earlier in the tree. For more information, see Defining what to do next.
- By configuring conditional responses to jump to other nodes. For more information, see Conditional responses.

- By configuring digression settings for dialog nodes. Digressions can also impact how users move through the nodes at run time. If you enable digressions away from most nodes and configure returns, users can jump from one node to another and back again more easily. For more information, see Digressions.

Sample dialog

This diagram shows a mockup of a dialog tree that is built with the graphical user interface dialog editor.



Dialog tree

The dialog tree in this diagram contains two root dialog nodes. A typical dialog tree would likely have many more nodes, but this depiction provides a glimpse of what a subset of nodes might look like.

- The first root node conditions on an intent value. It has two child nodes that each condition on an entity value. The second child node defines two responses. The first response is returned to the user if the value of the context variable matches the value that is specified in the condition. Otherwise, the second response is returned.

  This standard type of node is useful to capture questions about a certain topic and then in the root response ask a follow-up question that is addressed by the child nodes. For example, it might recognize a user question about discounts and ask a follow-up question about whether the user is a member of any associations with which the company has special discount arrangements. And the child nodes provide different responses based on the user's answer to the question about association membership.

- The second root node is a node with slots. It also conditions on an intent value. It defines a set of slots, one for each piece of information that you want to collect from the user. Each slot asks a question to elicit the answer from the user. It looks for a specific entity value in the user's reply to the prompt, which it then saves in a slot context variable.

  This type of node is useful for collecting details that you might need to perform a transaction on the user's behalf. For example, if the user's intent is to book a flight, the slots can collect the origin and destination location information, travel dates, and so on.

Creating a dialog

The dialog defines what your assistant says in response to customers.

The following nodes are created for you automatically:

- **Welcome**: The first node. It contains a greeting that is displayed to your users when they first engage with your assistant. You can edit the greeting.

  > **Note:** This node is not triggered in dialog flows that are initiated by users. For example, dialogs used in integrations with channels such as Facebook or Slack skip nodes with the `welcome` special condition.

- **Anything else**: The final node. It contains phrases that are used to reply to users when their input is not recognized. You can replace the responses that are provided or add more responses with a similar meaning to add variety to the conversation. You can also choose whether you want your assistant to return each response that is defined in turn or return them in random order.

For more information about these built-in nodes, see Starting and ending the dialog.

1. To add more nodes to the dialog tree, click **Add node**.

Your new node is added after the *Welcome* node and before the *Anything else* node.

2. Add a name to the node.

   Use a short, customer-friendly description of what the node does as its name. For example, `Open an account`, `Get policy information`, or `Get a weather forecast`.

   The name can be up to 512 characters in length.

   > **Tip:** This node name is shown to customers or service desk personnel to express the purpose of this branch of the dialog, so take some time to add a name that is concise and descriptive.

3. In the **If assistant recognizes** field, enter a condition that, when met, triggers your assistant to process the node.

   To start off, you typically want to add an intent as the condition. For example, if you add `#open_account` here, it means that you want the response that you will specify in this node to be returned to the user if the user input indicates that the user wants to open an account.

   As you begin to define a condition, a box is displayed that shows you your options. You can enter one of the following characters, and then pick a value from the list of options that is displayed.

Character	Lists defined values for these artifact types
#	intents
@	entities
@{entity-name}:	{entity-name} values
$	context-variables that you defined or referenced elsewhere in the dialog

Condition builder syntax

   You can create a new intent, entity, entity value, or context variable by defining a new condition that uses it. If you create an artifact this way, be sure to go back and complete any other steps that are necessary for the artifact to be created completely, such as defining sample utterances for an intent.

   To define a node that triggers based on more than one condition, enter one condition, and then click the plus sign (+) icon next to it. If you want to apply an `OR` operator to the multiple conditions instead of `AND`, click the `and` that is displayed between the fields to change the operator type. AND operations are executed before OR operations, but you can change the order by using parentheses. For example: `$isMember:true AND ($memberlevel:silver OR $memberlevel:gold)`

   The condition you define must be less than 2,048 characters in length.

   For more information about how to test for values in conditions, see Conditions.

4. **Optional**: If you want to collect multiple pieces of information from the user in this node, then click **Customize** and enable **Slots**. See Gathering information with slots for more details.

5. Enter a response.

   - Add the text or multimedia elements that you want your assistant to display to the user as a response.
   - If you want to define different responses based on certain conditions, then click **Customize** and enable **Multiple responses**.
   - For information about conditional responses, rich responses, or how to add variety to responses, see Responses.

6. Specify what to do after the current node is processed. You can choose from the following options:

   - **Wait for user input**: Your assistant pauses until new input is provided by the user.
   - **Skip user input**: Your assistant jumps directly to the first child node. This option is only available if the current node has at least one child node.
   - **Jump to**: Your assistant continues the dialog by processing the node you specify. You can choose whether your assistant should evaluate the target node's condition or skip directly to the target node's response. See Configuring the Jump to action for more details.

7. **Optional**: If you want this node to be considered when users are shown a set of node choices at run time, and asked to pick the one that best matches their goal, then add a short description of the user goal handled by this node to the **external node name** field. For example, *Open an account*.

   Plus The *external node name* field is only displayed only to users of paid plans. See Disambiguation for more details.

8. To add more nodes, select a node in the tree, and then click the **More** ⋮ icon.

    - To create a peer node that is checked next if the condition for the existing node is not met, select **Add node below**.
    - To create a peer node that is checked before the condition for the existing node is checked, select **Add node above**.
    - To create a child node to the selected node, select **Add child node**. A child node is processed after its parent node.
    - To copy the current node, select **Duplicate**.

    For more information about the order in which dialog nodes are processed, see Dialog overview.

9. Test the dialog as you build it.

    See Testing your dialog for more information.

Conditions

A node condition determines whether that node is used in the conversation. Response conditions determine which response to return to a user.

- Condition artifacts
- Special conditions
- Condition syntax details

For tips on performing more advanced tasks in conditions, see Condition usage tips.

Condition artifacts

You can use one or more of the following artifacts in any combination to define a condition:

- **Context variable**: The node is used if the context variable expression that you specify is true. Use the syntax, `$variable_name:value` or `$variable_name == 'value'`.

  For node conditions, this artifact type is typically used with an `AND` or `OR` operator and another condition value. That's because something in the user input must trigger the node; the context variable value being matched alone is not enough to trigger it. If the user input object sets the context variable value somehow, for example, then the node is triggered.

  > **Tip:** Do not define a node condition based on the value of a context variable in the same dialog node in which you set the context variable value.

  For response conditions, this artifact type can be used alone. You can change the response based on a specific context variable value. For example, `$city:Boston` checks whether the `$city` context variable contains the value, `Boston`. If so, the response is returned.

  For more information about context variables, see Context variables.

- **Entity**: The node is used when any value or synonym for the entity is recognized in the user input. Use the syntax, `@entity_name`. For example, `@city` checks whether any of the city names that are defined for the @city entity were detected in the user input. If so, the node or response is processed.

  > **Tip:** Consider creating a peer node to handle the case where none of the entity's values or synonyms are recognized.

  For more information about entities, see Defining entities.

- **Entity value**: The node is used if the entity value is detected in the user input. Use the syntax, `@entity_name:value` and specify a defined value for the entity, not a synonym. For example: `@city:Boston` checks whether the specific city name, `Boston`, was detected in the user input.

  > **Tip:** If you check for the presence of the entity, without specifying a particular value for it, in a peer node, be sure to position this node (which checks for a particular entity value) before the peer node that checks only for the presence of the entity. Otherwise, this node will never be evaluated.

  If the entity is a pattern entity with capture groups, then you can check for a certain group value match. For example, you can use the syntax: `@us_phone.groups[1] == '617'`

  See Storing and recognizing pattern entity groups in input for more information.

- **Intent**: The simplest condition is a single intent. The node is used if, after your assistant's natural language processing evaluates the user's input, it determines that the purpose of the user's input maps to the pre-defined intent. Use the syntax, `#intent_name`. For example, `#weather` checks if the user input is asking for a weather forecast. If so, the node with the `#weather` intent condition is processed.

  For more information about intents, see Defining intents.

- **Special condition**: Conditions that are provided with the product that you can use to perform common dialog functions. See the **Special conditions** table in the next section for details.

Special conditions

Condition syntax	Description
`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. The **Anything else** node is triggered by this condition. If you add search to your assistant, a root node with this condition can be configured to trigger a search.
`conversation_start`	Like **welcome**, this condition is evaluated as true during the first dialog turn. Unlike **welcome**, it is true whether or not the initial request from the application contains user input.
`false`	This condition is always evaluated to false. You might use this at the start 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.
`irrelevant`	This condition will evaluate to true if the user's input is determined to be irrelevant by the watsonx Assistant 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.
`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. The **Welcome** node is triggered by this condition. Typically, a node with this condition is used to greet the user, for example, to display a message such as `Welcome to our Pizza ordering app`. This node is never processed during interactions that occur through channels such as Facebook or Slack.

Special conditions

Condition syntax details

Use one of these syntax options to create valid expressions in conditions:

- Shorthand notations to refer to intents, entities, and context variables. See Accessing and evaluating objects.
- Spring Expression Language (SpEL), which is an expression language that supports querying and manipulating an object graph at run time. See Spring Expression Language (SpEL) for more information.

You can use regular expressions to check for values to condition against. To find a matching string, for example, you can use the `String.find` method. See Methods for more details.

Responses

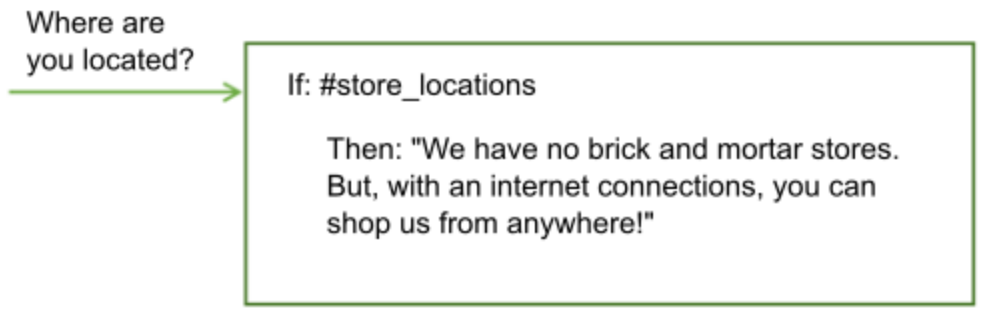
The dialog response defines how to reply to the user.

You can reply in the following ways:

- Simple text response
- Rich responses
- Conditional responses

Simple text response

If you want to provide a text response, simply enter the text that you want your assistant to display to the user.



{caption="Simple response" caption-side="bottom"}

To include a context variable value in the response, use the syntax `$variable_name` to specify it. See Context variables for more information. For example, if you know that the $user context variable is set to the current user's name before a node is processed, then you can refer to it in the text response of the node like this:

```
Hello $user
```

If the current user's name is `Norman`, then the response that is displayed to Norman is `Hello Norman`.

If you include one of these special characters in a text response, escape it by adding a backslash (`\`) in front of it. If you are using the JSON editor, you need to use two backslashes to escape (`\\`). Escaping the character prevents your assistant from misinterpreting it as being one of the following artifact types:

Special character	Artifact	Example
$	Context variable	The transaction fee is $2.
@	Entity	Send us your feedback at feedback@example.com.
#	Intent	We are the #1 seller of lobster rolls in Maine.

Special characters to escape in responses

The built-in integrations support the following Markdown syntax elements:

Format	Syntax	Example
Italics	We're talking about *practice*.	We're talking about *practice*.
Bold	There's **no** crying in baseball.	There's **no** crying in baseball.
Hypertext link	Contact us at [ibm.com](https://www.ibm.com).	Contact us at ibm.com.

Supported markdown syntax

If you don't code a link when you specify a phone number in a text response, it is not converted to a telephone link anywhere except in a web chat integration that is accessed from a mobile device.

The "Try it out" pane does not support Markdown syntax currently. For testing purposes, you can use the assistant *Preview* to see how the Markdown syntax is rendered.

The "Try it out" pane, assistant *Preview*, and *web chat* integration support HTML syntax. The *Slack* and *Facebook* integrations do not.

Learn more about simple responses

- Adding multiple lines
- Adding variety

Adding multiple lines

If you want a single text response to include multiple lines separated by carriage returns, then follow these steps:

1. Add each line that you want to show to the user as a separate sentence into its own response variation field. For example:

Response variations
Hi.
How are you today?

2. For the response variation setting, choose **multiline**.

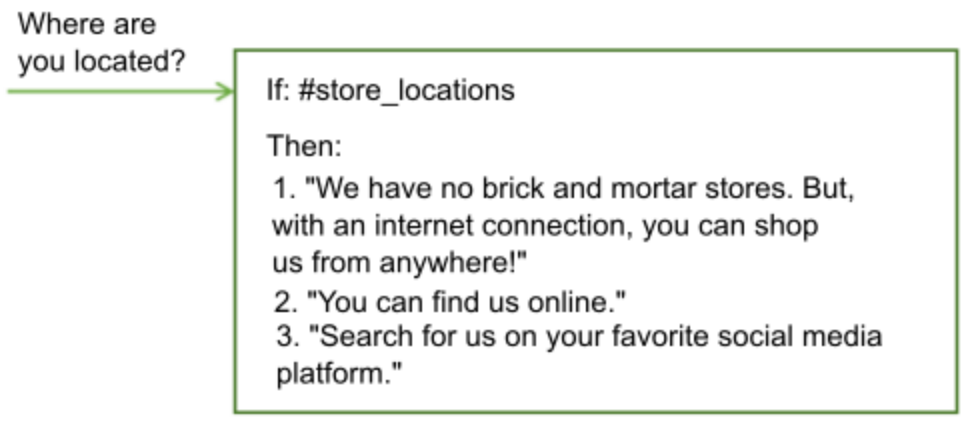
   When the response is shown to the user, both response variations are displayed, one on each line, like this:

   ```
   Hi.
   How are you today?
   ```

Adding variety

If your users return to your conversation service frequently, they might be bored to hear the same greetings and responses every time. You can add *variations* to your responses so that your conversation can respond to the same condition in different ways.

In this example, the answer that your assistant provides in response to questions about store locations differs from one interaction to the next:



{caption="Response variations" caption-side="bottom"}

You can choose to rotate through the response variations sequentially or in random order. By default, responses are rotated sequentially, as if they were chosen from an ordered list.

To change the sequence in which individual text responses are returned, complete the following steps:

1. Add each variation of the response into its own response variation field. For example:

Response variations
Hello.
Hi.
Howdy!

2. For the response variation setting, choose one of the following settings:

   - **sequential**: The system returns the first response variation the first time the dialog node is triggered, the second response variation the second time the node is triggered, and so on, in the same order as you define the variations in the node.

     Results in responses being returned in the following order when the node is processed:

     First time:

     ```
     Hello.
     ```

     Second time:

     ```
     Hi.
     ```

     Third time:

```
Howdy!
```

- **random**: The system randomly selects a text string from the variations list the first time the dialog node is triggered, and randomly selects another variation the next time, but without repeating the same text string consecutively.

  Example of the order that responses might be returned in when the node is processed:

  First time:

  ```
  Howdy!
  ```

  Second time:

  ```
  Hi.
  ```

  Third time:

  ```
  Hello.
  ```

Rich responses

You can return responses with multimedia or interactive elements such as images or clickable buttons to simplify the interaction model of your application and enhance the user experience.

In addition to the default response type of **Text**, for which you specify the text to return to the user as a response, the following response types are supported:

- **Connect to human agent** : The dialog calls a service that you designate, typically a service that manages human agent support ticket queues, to transfer the conversation to a person. You can optionally include a message that summarizes the user's issue to be provided to the human agent.
- **Channel transfer**: The dialog requests that the conversation be transferred to a different channel (for example, from the Slack integration to the web chat integration).
- **Image**: Embeds an image into the response. The source image file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Video**: Embeds a video player into the response. The source video must be hosted somewhere, either as a playable video on a supported video streaming service or as a video file with a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **Audio**: Embeds an audio clip into the response. The source audio file must be hosted somewhere and have a URL that you can use to reference it. It cannot be a file that is stored in a directory that is not publicly accessible.
- **iframe**: Embeds content from an external website, such as a form or other interactive component, directly within the chat. The source content must be publicly accessible using HTTP, and must be embeddable as an HTML `iframe` element.
- **Option**: Adds a list of one or more options. When a user clicks one of the options, an associated user input value is sent to your assistant. How options are rendered can differ depending on the number of options and where you deploy the dialog.
- **Pause**: Forces the application to wait for a specified number of milliseconds before continuing with processing. You can choose to show an indicator that the assistant is working on typing a response. Use this response type if you need to perform an action that might take some time.
- **Search skill**: Plus Searches an external data source for relevant information to return to the user. The data source that is searched is a Discovery service data collection that you configure when you add search to the assistant that uses this dialog.
- **User-defined**: If you use the JSON editor to define the response, you can create your own user-defined response type. For more information, see Defining responses using the JSON editor .

Different integrations have different capabilities for displaying rich responses. If you want to define different responses that are customized for different channels, you can do so by editing the response using the JSON editor. For more information, see Targeting specific integrations .

To add a rich response, complete the following steps:

1. Click the dropdown menu in the **Assistant responds** field to choose a response type, and then provide any required information.

   For more information, see the following sections:

   - **Connect to human agent**
   - **Channel transfer**
   - **Image**
   - **Option**

- Pause
- Search skill Plus
- Text

2. To add another response type to the current response, click **Add response type**.

   You might want to add multiple response types to a single response to provide a richer answer to a user query. For example, if a user asks for store locations, you could show a map and display a button for each store location that the user can click to get address details. To build that type of response, you can use a combination of image, options, and text response types. Another example is using a text response type before a pause response type so you can warn users before pausing the dialog.

   > **Note:** You cannot add more than 5 response types to a single response. This means that if you define three conditioned responses for a dialog node, each conditioned response can have no more than 5 response types added to it.

   > **Note:** You cannot add more than one **Connect to human agent** or more than one **Search skill** response type to a single dialog node.

   > **Note:** Do not add more than one option response type to a single dialog node because both lists are displayed at once, but the customer can choose an option from only one of them.

3. If you added more than one response type, you can click the **Move** up or down arrows to arrange the response types in the order you want your assistant to process them.

Adding a *Connect to human agent* response type

If your client application is able to transfer a conversation to a person, such as a customer support agent, then you can add a *Connect to human agent* response type to initiate the transfer. Some of the built-in integrations, such as web chat and Intercom, support making transfers to service desk agents. If you are using a custom application, you must program the application to recognize when this response type is triggered.

> **Tip:** If you want to take advantage of the *containment* metric to track your assistant's success rate, add this response type to your dialog or use an alternate method to identify when customers are directed to outside support. For more information, see Measuring containment.

To add a *Connect to human agent* response type, complete the following steps:

1. From the dialog node where you want to add the response type, click the dropdown menu in the **Assistant responds** field, and then choose **Connect to human agent**.
2. **Optional**. Add a message to share with the human agent to whom the conversation is transferred in the **Message to human agent** field.
3. Add a message to show to the customer to explain that they are being transferred.

   You can add a message to show when agents are available and a message to show when agents are unavailable. Each message can be up to 320 characters in length.

   Web chat built-in service desk integrations only: The text you add to the *Response when agents are online* and *Response when no agents are online* fields is used for transfers in web chat version 3 and later. If you don't add your own messages, the hint text (the grayed out text that is displayed as the example messages) is used.

   > **Tip:** If you use this response type in multiple nodes and want to use the same custom text each time, but don't want to have to edit each node individually, you can change the default text that is used by the web chat. To change the default messages, edit the language source file. Look for the `default_agent_availableMessage` and `default_agent_unavailableMessage` values. For more information about how to change web chat text, see Languages.

4. **Optional**: If the channel where you deploy the assistant is integrated with a service desk, you can add initial routing information to pass with the transfer request.
   - Pick the integration type from the **Service desk routing** field.
   - Add routing information that is meaningful to the service desk you are using.

Service desk type	Routing information	Description
Salesforce	Button ID	Specify a valid button ID from your Salesforce deployment.

Zendesk	Department	Specify a valid department name from your Zendesk account.

Service desk routing options

The dialog transfer does not occur when you test dialog nodes with this response type in the "Try it out" pane of the dialog. You must access a node that uses this response type from the *Preview* button for the assistant to see how your users will experience it.

Adding a *Channel transfer* response type

If your assistant uses multiple integrations to support different channels for interaction with users, there might be some situations when a customer begins a conversation in one channel but then needs to transfer to a different channel.

The most common such situation is transferring a conversation to the web chat integration, in order to take advantage of web chat features such as service desk integration.

Note: Currently, the web chat is the only supported target for a channel transfer.

Only the following integrations can initiate a channel transfer:

- Slack
- Facebook Messenger
- WhatsApp

Other integrations ignore the *Channel transfer* response type.

To add a *Channel transfer* response type, complete the following steps:

1. From the dialog node where you want to add the response type, click the dropdown menu in the **Assistant responds** field, and then choose **Channel transfer**.
2. **Optional**. In the **Message before link to web chat** field, edit the introductory message to display to the user (in the originating channel) before the link that initiates the transfer. By default, this message is `OK, click this link for additional help. Chat will continue on a new web page.`
3. In the **URL to web chat** field, type the URL for your website where the web chat widget is embedded.

In the integration that processes the *Channel transfer* response, the introductory message is displayed, followed by a link to the URL you specify. The user must then click the link to initiate the transfer.

When a conversation is transferred from one channel to another, the session history and context are preserved, so the destination channel can continue the conversation from where it left off. Note that the message output that contains the *Channel transfer* response is processed first by the channel that initiates the transfer, and then by the target channel. If the output contains multiple responses (perhaps using different response types), these will be processed by both channels (before and after the transfer). If you want to target individual responses to specific channels, you can do so by editing the response using the JSON editor. For more information, see Targeting specific integrations.

Adding an *Image* response type

Sometimes a picture is worth a thousand words. Include images in your response to do things like illustrate a concept, show off merchandise for sale, or maybe to show a map of your store location.

To add an *Image* response type, complete the following steps:

1. Choose **Image**.
2. Add the full URL to the hosted image file into the **Image source** field.

   The image must be in .jpg, .gif, or .png format. The image file must be stored in a location that is publicly addressable by an `https:` URL.

   For example: `https://www.example.com/assets/common/logo.png`.

   If you want to display an image title and description above the embedded image in the response, then add them in the fields provided.

   To access an image that is stored in IBM Cloud® Object Storage, enable public access to the individual image storage object, and then reference it by specifying the image source with syntax like this: `https://s3.eu.cloud-object-storage.appdomain.cloud/your-bucket-name/image-name.png`.

   Note: Some integration channels ignore titles or descriptions.

Adding a *Video* response type

Include videos in your response to share how-to demonstrations, promotional clips, and so forth. In the web chat, a video response renders as an embedded video player.

To add a *Video* response type, complete the following steps:

1. Choose **Video**.
2. Add the full URL to the hosted video into the **Video source** field:
   - To link directly to a video file, specify the URL to a file in any standard format such as MPEG or AVI. In the web chat, the linked video will render as an embedded video player.

     Note: HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.

   - To link to a video hosted on a supported video hosting service, specify the URL to the video. In the web chat, the linked video will render using the embeddable player for the hosting service.

     Note: Specify the URL you would use to view the video in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

     You can embed videos hosted on the following services:

     - YouTube
     - Facebook
     - Vimeo
     - Twitch
     - Streamable
     - Wistia
     - Vidyard

   If you want to display a video title and description above the embedded video in the response, then add them in the fields provided.

   Note: Some integration channels ignore titles or descriptions.

   If you want to scale the video to a specific display size, specify a number in the **Base height** field.
3. The **Video** response type is supported in the web chat, Facebook, WhatsApp, Slack, and SMS integrations.

Adding an *Audio* response type

Include audio clips in your response to share spoken-word or other audible content. In the web chat, a video response renders as an embedded video player. In the phone integration, an audio response plays over the phone.

To add an *Audio* response type, complete the following steps:

1. Choose **Audio**.
2. Add the full URL to the hosted audio clip into the **Audio source** field:
   - To link directly to an audio file, specify the URL to a file in any standard format such as MP3 or WAV. In the web chat, the linked audio clip will render as an embedded audio player.
   - To link to an audio clip on a supported audio hosting service, specify the URL to the audio clip. In the web chat, the linked audio clip will render using the embeddable player for the hosting service.

     Note: Specify the URL you would use to access the audio file in your browser (for example, `https://soundcloud.com/ibmresearch/fallen-star-amped`). You do not need to convert the URL to an embeddable form; the web chat will do this automatically.

     You can embed audio hosted on the following services:

     - SoundCloud
     - Mixcloud

   If you want to display a title and description above the embedded audio player in the response, then add them in the fields provided.

> **Note:** Some integration channels ignore titles or descriptions.

If you want the audio clip to loop indefinitely, select **On** in the **Loop** field. For example, you might want to use this option to play music while a user waits on the phone. (By default, the audio plays only once and then stops.)

> **Note:** The **Loop** option is currently supported only by the phone integration. This option has no effect if you are using the web chat integration or any other channel.

3. The **Audio** response type is supported in the web chat, Facebook, WhatsApp, Slack, SMS, and phone integrations.

Adding an *iframe* response

Add an *iframe* response to embed content from another website directly inside the chat window as an HTML `iframe` element. An iframe response is useful if you want to enable customers to perform some interaction with an external service without leaving the chat. For example, you might use an *iframe* response to display the following examples within the web chat:

- An interactive map on Google Maps
- A survey that uses SurveyMonkey
- A form for making reservations through OpenTable
- A scheduling form that uses Calendly

In the web chat, there are two ways the iframe can be included:

- Like a preview card that describes the embedded content. Customers can click this card to display the frame and interact with the content.

The *iframe* response type is supported by the following channel integrations:

- Web chat
- Facebook

To add an *iframe* response type, complete the following steps:

1. In the **Assistant says** field, click the **iframe** icon ( ).
2. Add the full URL to the external content in the **iframe source** field.

   The URL must specify content that is embeddable in an HTML `iframe` element. Different sites have different restrictions for embedding content, and different processes for generating embeddable URLs. An embeddable URL is one that can be specified as the value of the `src` attribute of the `iframe` element.

   For example, to embed an interactive map that uses Google Maps, you can use the Google Maps Embed API. For more information, see The Maps Embed API overview. Other sites have different processes for creating embeddable content.

   For the technical details of using `Content-Security-Policy: frame-src` that gives you permission to embed the website content in your assistant, see CSP: frame-src.
3. Optionally add a descriptive title in the **Title** field.

   In the web chat, the title that you add is displayed in the preview card. The customer clicks the preview card to render the external content.

> **Note:** If you do not specify a title, the web chat attempts to retrieve metadata from the specified URL and displays the content title per the specification in the source.

> **Note:** References to variables are not supported.

Technical details: `iframe` sandboxing

Content that is loaded in an iframe by the web chat is *sandboxed*, meaning that it restricts permissions that reduce security vulnerabilities. The web chat uses the `sandbox` attribute of the `iframe` element to grant only the following permissions:

Permission	Description
`allow-downloads`	Allows downloading files from the network, if the download is initiated by the user.

allow-forms	Allows submitting forms.
allow-scripts	Allows running scripts, but *not* opening pop-up windows.
allow-same-origin	Allows the content to access its own data storage (such as cookies), and allows limited access to JavaScript APIs.

Note: A script that runs inside a sandboxed iframe cannot change any content outside the iframe, *if* the outer page and the iframe have different origins. Be careful if you use an *iframe* response to embed content that has the same origin as the page where your web chat widget is hosted. In this situation the embedded content can defeat the sandboxing and gain access to content outside the frame. For more information about this potential vulnerability, see the `sandbox` attribute documentation.

Technical details: `iframe` preview card

The `iframe` response type in web chat displays the **Preview card**, which includes an image, title, and description of the webpage that the user visits in the web chat.

To display an image, title, and description in the **Preview card**, the webpage needs the following `<meta>` tags inside the `<head>` tag:

```
$ <meta property="og:image" content="https://.../image.jpg" />

<meta property="og:image:url" content="https://.../image.jpg" />

<meta property="og:title" content="The webpage title" />
<meta property="og:description" content="The webpage description" />
```

These metadata properties specified come from The Open Graph Protocol.

Tip: The metadata is optional. The web chat displays a preview card with the webpage url and metadata, that the web chat fetched successfully.

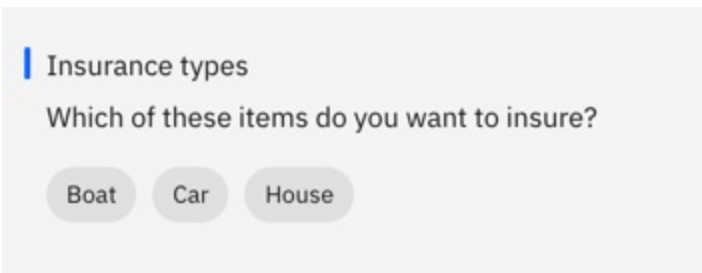
Adding an *Option* response type

Add an option response type when you want to give the customer a set of options to choose from. For example, you can construct a response like this:

List title	List description	Option label	User input submitted when clicked
Insurance types	Which of these items do you want to insure?		
		Boat	I want to buy boat insurance
		Car	I want to buy car insurance
		Home	I want to buy home insurance

Response options

Most integrations display the options as buttons if there are only a few items (4 or fewer, for example).



{caption="Option buttons" caption-side="bottom"}

Otherwise, the options are displayed as a list.

To add an *Option* response type, complete the following steps:

1. From the dialog node where you want to add the response type, click the dropdown menu in the **Assistant responds** field, and then choose **Option**.
2. Click **Add option**.

3. In the **List label** field, enter the option to display in the list.

   The label must be less than 2,048 characters in length.

4. In the corresponding **Value** field, enter the user input to pass to your assistant when this option is selected.

   The value must be less than 2,048 characters in length.

   > ⚠ **Important:** For Slack integrations where the options are displayed as a list, each value must be 75 characters or less in length.

   Specify a value that you know will trigger the correct intent when it is submitted. For example, it might be a user example from the training data for the intent.

5. Repeat the previous steps to add more options to the list.

   You can add up to 20 options.

6. Add a list introduction in the **Title** field. The title can ask the user to pick from the list of options.

   > **Note:** Some integration channels do not display the title.

7. Optionally, add additional information in the **Description** field. If specified, the description is displayed after the title and before the option list.

   > **Note:** Some integration channels do not display the description.

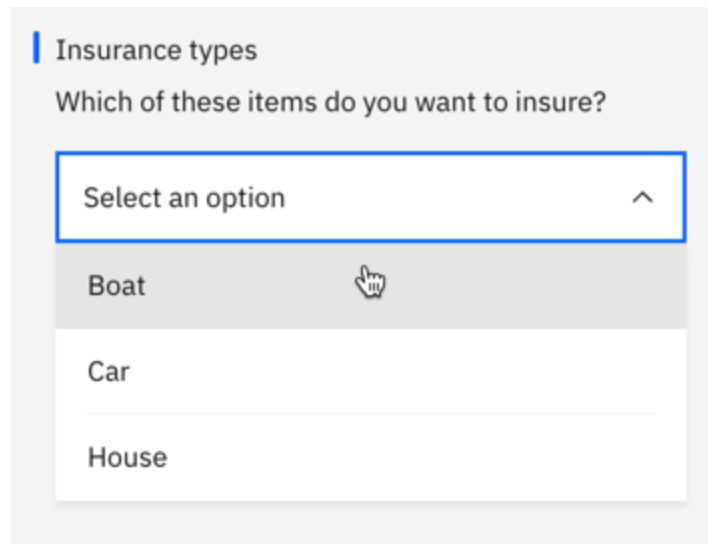
8. **Optional**: If you want to indicate a preference for how the options are displayed, as buttons or in a list, you can add a `preference` property for the response.

   To do so, open the JSON editor for the response, and then add a `preference` name and value pair before the `response_type` name and value pair. You can set the preference to `dropdown` or `button`.

```
{
  "output": {
    "generic": [
      {
        "title": "Insurance types",
        "options": [
          {
            "label": "Boat",
            "value": {
              "input": {
                "text": "I want to buy boat insurance."
              }
            }
          },
          {
            "label": "Car",
            "value": {
              "input": {
                "text": "I want to buy car insurance."
              }
            }
          },
          {
            "label": "House",
            "value": {
              "input": {
                "text": "I want to buy house insurance."
              }
            }
          }
        ],
        "preference": "dropdown", //add this name and value pair
        "description": "Which of these items do you want to insure?",
        "response_type": "option"
      }
    ]
}
```

```
}
```

When you define an options list with only 3 items, the options are typically displayed as buttons. When you add a preference property that indicates `dropdown` as the preference, for example, you can see in the "Try it out" pane that the list is displayed as a drop-down list instead.



Options list

Some integration types, such as the web chat, reflect your preference. Other integration types, such as Slack, do not reflect your preference when they render the options.

> ⚠ **Important:** Do not add more than one option response type to a single dialog node because both lists are displayed at once, but the customer can choose an option from only one of them.

If you need to be able to populate the list of options with different values based on some other factors, you can design a dynamic options list. For more information, see the How to Dynamically Add Response Options to Dialog Nodes blog post.

Adding a *Pause* response type

Add a pause response type to give the assistant time to respond. For example, you might add a pause response type to a node that calls a webhook. The pause indicates that the assistant is working on an answer, which gives the assistant time to make the webhook call and get a response. Then, you can jump to a child node to show the result.

To add a *Pause* response type, complete the following steps:

1. From the dialog node where you want to add the response type, click the dropdown menu in the **Assistant responds** field, and then choose **Pause**.
2. Add the length of time for the pause to last as a number of milliseconds (ms) to the **Duration** field.

   The value cannot exceed 10,000 ms. Users are typically willing to wait about 8 seconds (8,000 ms) for someone to enter a response. To prevent a typing indicator from being displayed during the pause, choose **Off**.

   > **Tip:** Add another response type, such as a text response type, after the pause to clearly denote that the pause is over.

This response type does not render in the "Try it out" pane. You must access a node that uses this response type from a test deployment to see how your users will experience it.

Adding a *Search skill* response type

Plus

If you have existing customer-facing material, such as an FAQ, a product catalog, or sales material that can answer questions that customers often ask, put that information to use. You can trigger a search of the existing material in real time to get the latest and most up-to-date answer for your customers.

To use the search skill response type, you must add search to the same assistant that uses this dialog. For more information, see IBM Watson® Discovery search integration setup.

To add a *Search skill* response type, complete the following steps:

1. From the dialog node where you want to add the response type, click the dropdown menu in the **Assistant responds** field, and then choose **Search skill**.

   Indicates that you want to search an external data source for a relevant response.

2. To edit the search query to pass to the Discovery service, click **Customize**, and then fill in the following fields:

- **Query**: Optional. You can specify a specific query in natural language to pass to Discovery. If you do not add a query, then the customer's exact input text is passed as the query.

  For example, you can specify `What cities do you fly to?`. This query value is passed to Discovery as a search query. Discovery uses natural language understanding to understand the query and to find an answer or relevant information about the subject in the data collection that is configured for the search.

  You can include specific information provided by the user by referencing entities that were detected in the user's input as part of the query. For example, `Tell me about @product`. Or you can reference a context variable, such as `Do you have flights to $destination?`. Just be sure to design your dialog such that the search is not triggered unless any entities or context variables that you reference in the query have been set to valid values.

  This field is equivalent to the Discovery `natural_language_query` parameter. For more information, see Query parameters.

  - **Filter**: Optional. Specify a text string that defines information that must be present in any of the search results that are returned.

    - To indicate that you want to return only documents with positive sentiment detected, for example, specify `enriched_text.sentiment.document.label:positive`.
    - To filter results to includes only documents that the ingestion process identified as containing the entity `Boston, MA`, specify `enriched_text.entities.text:"Boston, MA"`.
    - To filter results to includes only documents that the ingestion process identified as containing a product name provided by the customer, you can specify `enriched_text.entities.text:@product`.
    - To filter results to includes only documents that the ingestion process identified as containing a city name that you saved in a context variable named `$destination`, you can specify `enriched_text.entities.text:$destination`.

    This field is equivalent to the Discovery `filter` parameter. For more information, see Query parameters.

  If you add both a query and a filter value, the filter parameter is applied first to filter the data collection documents and cache the results. The query parameter then ranks the cached results.

3. **Optional**: Change the query type that is used for the search.

   The search sends a natural language query to Discovery automatically. If you want to use the Discovery query language instead, you can specify it. To do so, open the JSON editor for the node response.

   Edit the JSON code snippet to replace `natural_language` with `discovery_query_language`. For example:

```
{
  "output": {
    "generic": [
      {
        "query": "",
        "filter": "enriched_text.sentiment.document.label:positive",
        "query_type": "discovery_query_language",
        "response_type": "search_skill"
      }
    ]
  }
}
```

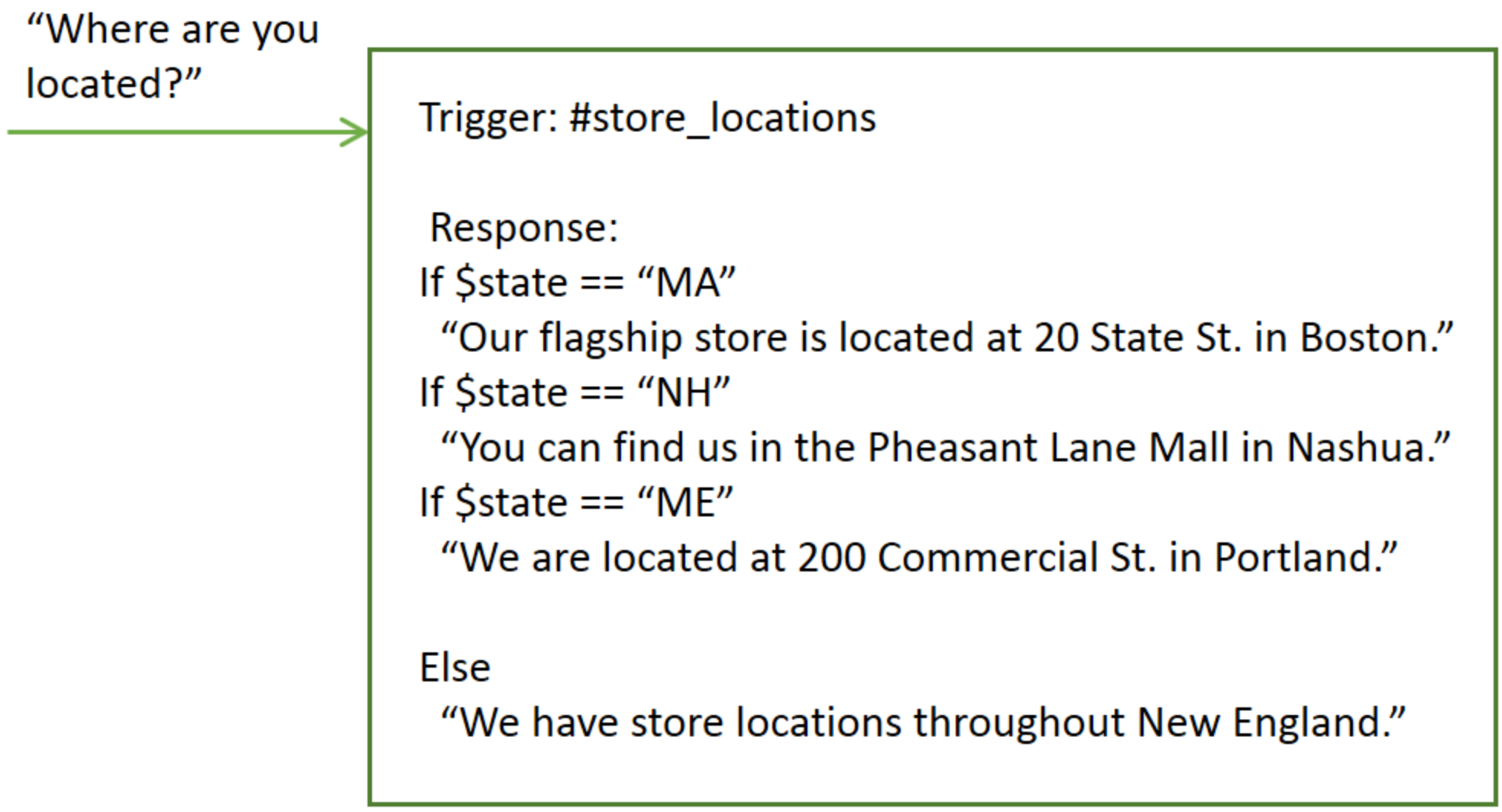
Test this response type from the assistant *Preview*. You cannot test it from the "Try it out" pane.

Conditional responses

A single dialog node can provide different responses, each one triggered by a different condition. Use this approach to address multiple scenarios in a single node.

The node still has a main condition, which is the condition for using the node and processing the conditions and responses that it contains.

In this example, your assistant uses information that it collected earlier about the user's location to tailor its response, and provide information about the store nearest the user. See Context variables for more information about how to store information collected from the user.



{caption="Conditional responses" caption-side="bottom"}

This single node now provides the equivalent function of four separate nodes.

To add conditional responses to a node, complete the following steps:

1. Click **Customize**, and then set the **Multiple conditioned responses** switch to **On**.

   The node response section changes to show a pair of condition and response fields. You can add a condition and a response into them.

2. To customize a response further, click the **Customize response** icon next to the response.

   You must open the response for editing to complete the following tasks:

   - **Update context**. To change the value of a context variable when the response is triggered, specify the context value in the context editor. You update context for each individual conditional response; there is no common context editor or JSON editor for all conditional responses.
   - **Add rich responses**. To add more than one text response or to add response types other than text responses to a single conditional response, you must open the edit response view.
   - **Configure a jump**. To instruct your assistant to jump to a different node after this conditional response is processed, select **Jump to** from the *And finally* section of the response edit view. Identify the node that you want your assistant to process next. See Configuring the Jump to action for more information.

     A **Jump to** action that is configured for the node is not processed until all of the conditional responses are processed. Therefore, if a conditional response is configured to jump to another node, and the conditional response is triggered, then the jump configured for the node is never processed, and so does not occur.

3. Click **Add response** to add another conditional response.

The conditions within a node are evaluated in order, just as nodes are. Be sure that your conditional responses are listed in the correct order. If you need to change the order, select a condition and response pair and move it up or down in the list using the arrows that are displayed.

Defining what to do next

After making the specified response, you can instruct your assistant to do one of the following things:

- **Wait for user input**: Your assistant waits for the user to provide new input that the response elicits. For example, the response might ask the user a yes or no question. The dialog will not progress until the user provides more input.
- **Skip user input**: Use this option when you want to bypass waiting for user input and go directly to the first child node of the current node instead.

> **Note:** The current node must have at least one child node for this option to be available.

- **Jump to another dialog node** : Use this option when you want the conversation to go directly to an entirely different dialog node. You can use a *Jump to* action to route the flow to a common dialog node from multiple locations in the tree, for example.

  > **Note:** The target node that you want to jump to must exist before you can configure the jump to action to use it.

Configuring the Jump to action

If you choose to jump to another node, specify when the target node is processed by choosing one of the following options:

- **Condition**: If the statement targets the condition section of the selected dialog node, your assistant checks first whether the condition of the targeted node evaluates to true.
  - If the condition evaluates to true, the system processes the target node immediately.
  - If the condition does not evaluate to true, the system moves to the next sibling node of the target node to evaluate its condition, and repeats this process until it finds a dialog node with a condition that evaluates to true.
  - If the system processes all the siblings and none of the conditions evaluate to true, the basic fallback strategy is used, and the dialog evaluates the nodes at the base level of the dialog tree.

  Targeting the condition is useful for chaining the conditions of dialog nodes. For example, you might want to first check whether the input contains an intent, such as `#turn_on`, and if it does, you might want to check whether the input contains entities, such as `@lights`, `@radio`, or `@wipers`. Chaining conditions helps to structure larger dialog trees.

  > **Note:** Avoid choosing this option when configuring a jump-to from a conditional response that goes to a node situated above the current node in the dialog tree. Otherwise, you can create an infinite loop. If your assistant jumps to the earlier node and checks its condition, it is likely to return false because the same user input is being evaluated that triggered the current node last time through the dialog. Your assistant will go to the next sibling or back to root to check the conditions on those nodes, and will likely end up triggering this node again, which means the process will repeat itself.

- **Response**: If the statement targets the response section of the selected dialog node, it is run immediately. That is, the system does not evaluate the condition of the selected dialog node; it processes the response of the selected dialog node immediately.

  Targeting the response is useful for chaining several dialog nodes together. The response is processed as if the condition of this dialog node is true. If the selected dialog node has another **Jump to** action, that action is run immediately, too.

- **Wait for user input** : Waits for new input from the user, and then begins to process it from the node that you jump to. This option is useful if the source node asks a question, for example, and you want to jump to a separate node to process the user's answer to the question.

Next steps

- Be sure to test your dialog as you build it. For more information, see Testing the dialog.
- For more information about ways to address common use cases, see Dialog building tips.
- For more information about the expression language that you can use to improve your dialog, such as methods that reformat dates or text, see Expression language methods.

You can also use the API to add nodes or otherwise edit a dialog. See Modifying a dialog using the API for more information.

Starting and ending the dialog

Learn more about how to use the nodes that are added to your dialog automatically to start and end the conversation.

The following dialog nodes are included automatically:

- **Welcome**: Defines how the assistant greets the user and starts the conversation.
- **Anything else**: What the assistant says when a customer's request cannot be satisfied by any of the defined intents.

Starting the conversation

The Welcome node is defined by using the `welcome` special condition, which is triggered when the assistant, rather than the user, starts the conversation. This happens when the integration or client application starts the session with an empty message and then waits for the assistant to greet the user, as in the following situations:

- *Preview* for the assistant
- "Try it out" pane

- *Web chat* integration with home screen disabled

However, the Welcome node is skipped in situations when the user initiates the conversation by sending a message, such as with the *Slack* and *Facebook* integrations. It is also skipped when the *Web chat* integration is used with the home screen enabled because in this situation the home screen provides the greeting. The home screen is enabled by default.

Unlike the `welcome` special condition, the `conversation_start` special condition is always triggered at the start of a conversation. You can use a combination of nodes with these two special conditions (`welcome` and `conversation_start`) to manage the start of your dialog in a consistent way.

For more information, see Special conditions.

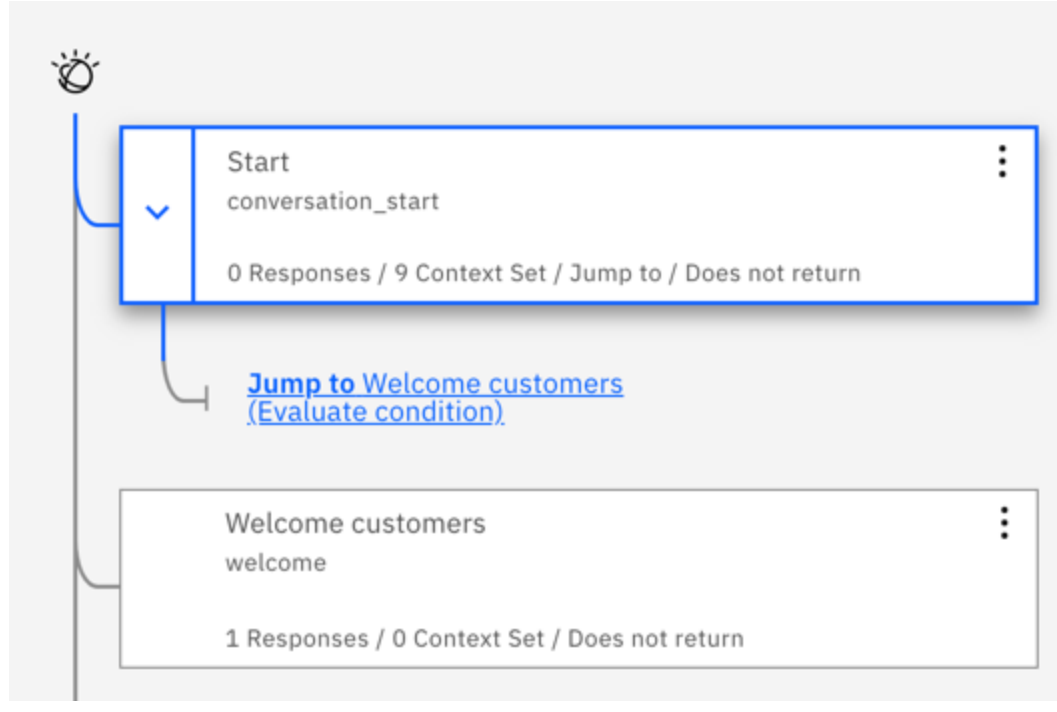
Setting initial context

If you need to set initial context variables at the beginning of each conversation, make sure you do so in a way that works with all of the integrations you plan to use. Do not use the Welcome node to set initial context variables unless you are certain your dialog isaccessed only in situations where the `welcome` special condition is triggered.

A safer and more consistent approach is to always set any initial context in a node that is defined by using the `conversation_start` special condition, which is always triggered. You can use this node in addition to a Welcome node that displays a greeting.

To manage the start of any conversation regardless of integration, follow these steps:

1. Add a dialog node above the Welcome node that is automatically added to the top of the dialog tree when you create the dialog.
2. Set the node condition for this newly added node to `conversation_start`. This node is reliably triggered at the beginning of any conversation.
3. In the `conversation_start` node, define any default values for context variables, and call any webhooks that you need to call at the beginning of every conversation.
4. Do not define a text response for this node. Instead, configure this node to jump to the `Welcome` node directly below it in the dialog tree (or whichever other node you want to process first), and choose **If assistant recognizes (condition)**.



Dialog tree

This design results in a dialog that works like this:

- Whatever the integration type, the `conversation_start` node is processed, which means any context variables that you define in it are initialized.
- In integrations where the assistant starts the dialog flow, the `Welcome` node is triggered and its text response is displayed.
- In integrations where the user starts the dialog flow, the user's first input is evaluated and then processed by the node that can provide the best response.

Ending the conversation gracefully

The *Anything else* node is designed to recognize the `anything_else` special condition, which understands when user input does not match any of the intents that are used as conditions in a dialog's nodes.

Don't delete the *Anything else* node.

You might not recognize its value at first, but it serves some important functions. If you do delete it, don't panic. You can add it back as a dialog node at the end of your dialog tree, and add the `anything_else` special condition to its *If assistant recognizes* field.

The *Anything else* node provides the following benefits:

- It prevents your assistant from ever going silent and failing to respond at all to your customers. The *Anything else* node is what enables your assistant

to (if nothing else) say, `I'm sorry, I didn't understand.` or `I can't help you with that.`

- Analytics uses this node to learn about the topics that your dialog can't address. The *coverage metric* looks for occurrences of nodes with the `anything_else` condition processed in the user conversation logs. It uses this information to determine the frequency with which your dialog is able to match user requests to intents that can address them. The node is registered by the metric if it conditions on `anything_else` alone or when used in combination with another condition, such as `anything_else && #positive_feedback`.
- If you want your assistant to redirect queries to the search integration when the dialog is unable to address them, this node recognizes when it's time to initiate the search. It's when a customer's message reaches the `anything_else` node that the message is sent to the search integration to find a relevant answer in your configured data collections. For more information, see Search trigger.

> **Note:** Messages that trigger search in this way are still registered by the coverage metric as messages that are *not* covered.

Connecting customers with support

No matter where you deploy your assistant, give your customers a way to get more support when they need it.

Build your dialog to recognize when customers need help that cannot be provided by the assistant. Add logic that can connect your customers to whatever type of professional support you offer. Your solutions might include:

- A toll-free phone number to a call center that is staffed by live agents
- An online support ticket form that customers complete and submit
- A service desk solution that is configured to work with your web chat integration or a custom client application

Design your dialog to recognize customer requests for help and address them. Add an intent that understands the customer request, and then add a dialog branch that handles the request.

For example, you might add an intent and use it in a dialog node like these intents:

Intent name	Intent user example 1	Intent user example 2	Response from dialog node that conditions on intent
#call_support	*How do I reach support?*	*What's your toll-free number?*	*Call 1-800-555-0123 to reach a call center agent at any time.*
#support_ticket	*How do I get help?*	*Who can help me with an issue I'm having?*	*Go to [Support Center](https://example.com/support) and open a support ticket.*

Alternative support request intent examples

If you deploy your assistant with an integration that has built-in service desk support, you can initiate a transfer by using the special *Connect to human agent* response type in your dialog response.

Adding chat transfer support

Design your dialog so that it can transfer customers to live agents. Consider adding support for initiating a transfer in the following scenarios:

- Anytime a user asks to speak to a person.

  Create an intent that can recognize when a customer asks to speak to someone. After you define the intent, you can add a root-level dialog node that conditions on the intent. As the dialog node response, add a *Connect to human agent* response type. At run time, if the user asks to speak to someone, this node is triggered and a transfer is initiated on the user's behalf.

- When the conversation broaches a topic that is sensitive in nature, you can start a transfer.

  For example, an insurance company might want questions about bereavement benefits always to be handled by a person. Or, if a customer wants to close an account, you might want to transfer the conversation to a representative who is authorized to offer incentives to keep the customer's business.

- When the assistant repeatedly fails to understand a customer's request. Instead of asking the customer to retry or rephrase the question over and over, your assistant can offer to transfer the customer to a live agent.

To design a dialog that can transfer the conversation, complete the following steps:

1. Add an intent that can recognize a user's request to speak to a live agent.

   You can create your own intent or add the prebuilt intent `#General_Connect_to_Agent` that is provided with the **General** content catalog.

2. Add a root node to your dialog that conditions on the intent you created in the previous step. Choose **Connect to human agent** as the response type.

   For more information, see Adding a *Connect to human agent* response type.

3. Add meaningful names to the dialog nodes in your dialog.

   When a conversation is transferred to an agent, the name of the most-recently processed dialog node is sent with it. To ensure that a useful summary is provided to service desk agents when a conversation is transferred to them, add short dialog node names that reflect the node's purpose. For example, *Find a store*.

   > **Note:** Every dialog branch can be processed by the assistant while it chats with a customer, including branches with root nodes in folders.

4. If a child node in any dialog branch conditions on a request or question that you do not want the assistant to handle, add a **Connect to human agent** response type to the child node.

   At run time, if the conversation reaches this child node, the dialog is passed to a live agent at that point.

Your dialog is now ready to support transfers from your assistant to a service desk agent.

Measuring containment

From the Analytics page, you can measure conversation containment. Containment is the rate at which your assistant is able to satisfy a customer's request without live agent intervention per conversation.

To measure containment accurately, the metric must be able to identify when a live agent intervention occurs. The metric primarily uses the *Connect to human agent* response type as an indicator. If a user conversation log includes a call to a *Connect to human agent* response type, then the conversation is considered to be *not contained*.

However, not all live agent interventions are transacted by using a *Connect to human agent* response type. If you use an alternative method to deliver more support, you must take extra steps to register the fact that the customer's need was not fulfilled by the assistant. For example, you might direct customers to your call center phone number or to an online support ticket form URL. If so, you need to flag these types of redirects so the containment metric is able to pick them up.

To enable the containment metric, complete the following steps:

1. For any dialog nodes that transfer the chat to a service desk agent, add a *Connect to human agent* response type.

   For more information, see Adding a *Connect to human agent* response type.

2. For any dialog nodes that redirect customers to alternative forms of customer support, add a context variable that is named `context_connect_to_agent` and set it to `true`.

   The process that calculates the containment metric looks for instances of this context variable, and registers any occurrences that it finds as interventions.

   - From the dialog node, go to the *Assistant responds* section and click the menu icon ⋮.

     > **Tip:** If you're using multiple conditioned responses, you must click **Customize response** to see the menu that is associated with the response.

   - Click **Open context editor**.
   - Add the following variable name and value to define the context variable:

Variable	Value
`context_connect_to_agent`	`true`

     Special containment context variable

   - Click **X** to close the dialog node. Your change is saved automatically.

     > **Tip:** If you're using multiple conditioned responses, you must click **Save** first.

Conversation building tips

Get tips about ways to address common tasks.

Adding nodes

- Add a node name that describes the purpose of the node.

  Today, you know what the node does, but months from now you might not remember. And the node name is displayed in the log, which can help you debug a conversation later.

- To gather the information that is required to complete a task, use a node with slots instead of a bunch of separate nodes to elicit information from users. See Gathering information with slots.

- For a complex process flow, tell users about any information they need to provide at the start of the process.

- Understand how your assistant travels through the dialog tree and the impact that folders, branches, jump-tos, and digressions have on the route. See Dialog flow.

- Do not add jump-tos everywhere. They increase the complexity of the dialog flow, and make it harder to debug the dialog later.

- To jump to a node in the same branch as the current node, use *Skip user input* instead of a *Jump-to*.

  This choice prevents you from having to edit the current node's settings when you remove or reorder the child nodes that are jumped to. See Defining what to do next.

- Before you enable digressions away from a node, test the most common user scenarios. And be sure that likely digressed-to nodes are configured to return. See Digressions.

Adding responses

- Keep answers short and useful.

- Reflect the user's intent in the response.

  Doing so assures users that the bot is understanding them, or if it is not, gives users a chance to correct a misunderstanding immediately.

- Include links to external sites in responses if the answer depends on data that changes frequently.

- Avoid overusing buttons. Encouraging users to pick predefined options from a set of buttons is less like a real conversation, and decreases your ability to learn what users really want to do. When you let real users ask for things in their own words, you can use the input to train the system and derive better intents.

- Avoid using a bunch of nodes when one node is sufficient. For example, add multiple conditional responses to a single node to return different responses depending on details provided by the user. See Conditional responses.

- Word your responses carefully. You can change how someone reacts to your system based simply on how you phrase a response. Changing one line of text can prevent you from having to write multiple lines of code to implement a complex programmatic solution.

Tips for capturing information from user input

It can be difficult to know the syntax to use in your dialog node to accurately capture the information you want to find in the user input. Here are some approaches that you can use to address common goals.

- **Returning the user's input**: You can capture the exact text uttered by the user and return it in your response. Use the following SpEL expression in a response to repeat the text that the user specified back in the response:

  ```
  You said: <? input.text ?>.
  ```

  > **Note:** If autocorrection is on, and you want to return the user's original input before it was corrected, you can use `<? input.original_text ?>`. But, be sure to use a response condition that checks whether the `original_text` field exists first.

- **Determining the number of words in user input** : You can perform any of the supported String methods on the input.text object. For example, you can find out how many words there are in a user utterance by using the following SpEL expression:

  ```
  input.text.split(' ').size()
  ```

- **Dealing with multiple intents**: A user enters input that expresses two separate tasks. `I want to open a savings account and apply for a credit card.` How does the dialog recognize and address both of them? See the Compound questions blog entry.

- **Dealing with ambiguous intents**: A user enters input that is ambiguous enough that your assistant finds two or more nodes with intents that might potentially address it. How does the dialog know which dialog branch to follow? If you enable disambiguation, it can show users their options and ask the user to pick the correct one. See Disambiguation for more details.

- **Handling multiple entities in input**: If you want to evaluate only the value of the first detected instance of an entity type, you can use the syntax `@entity == 'specific-value'` instead of the `@entity:(specific-value)` format.

  For example, when you use `@appliance == 'air conditioner'`, you are evaluating only the value of the first detected `@appliance` entity. But, using `@appliance:(air conditioner)` gets expanded to `entity['appliance'].contains('air conditioner')`, which matches whenever there is at least one `@appliance` entity of value 'air conditioner' detected in the user input.

- **Hiding data from the log**: You can prevent information from being stored in logs by storing it in a context variable and nesting the context variable within the `$private` section of the message context. For example, `$private.my_info`. Storing data in the private object hides it from the logs only. The information is still stored in the underlying JSON object. Do not allow this information to be exposed to the client application.

- **Checking for personal information**: If you want to check for and prevent a user from submitting personally identifiable information (PII) to some later process, you can add a dialog node that conditions on a pattern entity. Place the node at the beginning of the dialog tree to ensure that it checks the input first. For example, the entity might check for US Social Security number patterns or email address patterns. It can then respond with something like, `Please do not submit personally identifiable information. Can you reenter your request?` You can optionally reset the context to ensure that the user-submitted information with PII is not retained.

Condition usage tips

- **Checking for values with special characters**: If you want to check whether an entity or context variable contains a value, and the value includes a special character, such as an apostrophe ('), then you must surround the value that you want to check with parentheses. For example, to check if an entity or context variable contains the name `O'Reilly`, you must surround the name with parentheses.

  `@person:(O'Reilly)` and `$person:(O'Reilly)`

  Your assistant converts these shorthand references into these full SpEL expressions:

  `entities['person']?.contains('O''Reilly')` and `context['person'] == 'O''Reilly'`

  > **Note:** SpEL uses a second apostrophe to escape the single apostrophe in the name.

- **Checking for multiple values**: If you want to check for more than one value, you can create a condition that uses OR operators (`||`) to list multiple values in the condition. For example, to define a condition that is true if the context variable `$state` contains the abbreviations for Massachusetts, Maine, or New Hampshire, you can use this expression:

  `$state:MA || $state:ME || $state:NH`

- **Checking for number values**: When you compare numbers, first make sure the entity or variable that you are checking has a value. If the entity or variable does not have a number value, it is treated as having a null value (0) in a number comparison.

  For example, you want to check whether a dollar value included in user input is less than 100. If you use the condition `@price < 100`, and the `@price` entity is null, then the condition is evaluated as `true` because 0 is less than 100, even though the price was never set. To prevent this type of inaccurate result, use a condition such as `@price AND @price < 100`. If `@price` has no value, then this condition correctly returns false.

- **Checking for intents with a specific intent name pattern**: You can use a condition that looks for intents that match a pattern. For example, to find any detected intents with intent names that start with 'User_', you can use a syntax like this in the condition:

  `intents[0].intent.startsWith("User_")`

  However, when you do so, all of the detected intents are considered, even those with a confidence lower than 0.2. Also, check that intents that are considered irrelevant based on their confidence score are not returned. To do so, change the condition as follows:

  `!irrelevant && intents[0].intent.startsWith("User_")`

- **How fuzzy matching impacts entity recognition**: If you use an entity as the condition and fuzzy matching is enabled, then `@entity_name` evaluates to true only if the confidence of the match is greater than 30%. That is, only if `@entity_name.confidence > .3`.

Storing and recognizing entity pattern groups in input

To store the value of a pattern entity in a context variable, append `.literal` to the entity name. Using this syntax ensures that the exact span of text from user input that matched the specified pattern is stored in the variable.

Variable	Value
email	`<? @email.literal ?>`

Append .literal

To store the text from a single group in a pattern entity with groups defined, specify the array number of the group that you want to store. For example, assume that the entity pattern is defined as follows for the @phone_number entity. (Remember, the parentheses denote pattern groups):

```
\b((958)|(555))-(\d{3})-(\d{4})\b
```

To store only the area code from the phone number that is specified in user input, you can use the following syntax:

Variable	Value
area_code	`<? @phone_number.groups[1] ?>`

Area code only

The groups are delimited by the regular expression that is used to define the group pattern. For example, if the user input that matches the pattern that is defined in the entity `@phone_number` is: `958-234-3456`, then the following groups are created:

Group number	Regex engine value	Dialog value	Explanation
groups[0]	958-234-3456	958-234-3456	The first group is always the full matching string.
groups[1]	((958)|(555))	958	String that matches the regex for the first defined group, which is ((958)|(555)).
groups[2]	(958)	958	Match against the group that is included as the first operand in the OR expression ((958)|(555))
groups[3]	(555)	null	No match against the group that is included as the second operand in the OR expression ((958)|(555))
groups[4]	(\d{3})	234	String that matches the regular expression that is defined for the group.
groups[5]	(\d{4})	3456	String that matches the regular expression that is defined for the group.

Group details

To help you decipher which group number to use to capture the section of input you are interested in, you can extract information about all the groups at once. Use the following syntax to create a context variable that returns an array of all the grouped pattern entity matches:

Variable	Value
array_of_matched_groups	`<? @phone_number.groups ?>`

Array of matched groups

Use the "Try it out" pane to enter some test phone number values. For the input `958-123-2345`, this expression sets `$array_of_matched_groups` to `["958-123-2345","958","958",null,"123","2345"]`.

You can then count each value in the array starting with 0 to get the group number for it.

Array element value	Array element number
"958-123-2345"	0
"958"	1
"958"	2
null	3
"123"	4
"2345"	5

Array elements

From the result, you can determine that to capture the last four digits of the phone number, you need group #5, for example.

To return the JSONArray structure that is created to represent the grouped pattern entity, use the following syntax:

Variable	Value
json_matched_groups	`<? @phone_number.groups_json ?>`

JSON of matched groups

This expression sets `$json_matched_groups` to the following JSON array:

```
[
  {"group": "group_0","location": [0, 12]},
  {"group": "group_1","location": [0, 3]},
  {"group": "group_2","location": [0, 3]},
  {"group": "group_3"},
  {"group": "group_4","location": [4, 7]},
  {"group": "group_5","location": [8, 12]}
]
```

Note: `location` is a property of an entity that uses a zero-based character offset to indicate where the detected entity value begins and ends in the input text.

If you expect two phone numbers to be supplied in the input, then you can check for two phone numbers. If present, use the following syntax to capture the area code of the second number, for example.

Variable	Value
second_areacode	`<? entities['phone_number'][1].groups[1] ?>`

JSON of matched groups

If the input is `I want to change my phone number from 958-234-3456 to 555-456-5678`, then `$second_areacode` equals `555`.

Personalizing the dialog with context

To personalize the conversation, your assistant can collect information from the customer and then refer to it later in the conversation.

Retaining information across dialog turns

The dialog is stateless, meaning that it does not retain information from one interaction with the user to the next. When you add a dialog to an assistant and deploy it, the assistant saves the context from one message call, and then resubmits it on the next request throughout the current session. The current session lasts while a user interacts with the assistant plus the designated session inactivity time frame. The maximum session inactivity time allowed ranges from 5 minutes to 7 days, depending on your plan type. If you do not add the dialog to an assistant, it is your responsibility as the custom application developer to maintain any continuing information that the application needs.

The application can pass information to the dialog, and the dialog can update this information and pass it back to the application, or to a subsequent node. The dialog does so by using *context variables*.

Context variables

A context variable is a variable that you define in a node. You can specify a default value for it. Then, other nodes, application logic, or user input can set or change the value of the context variable.

You can condition against context variable values by referencing a context variable from a dialog node condition to determine whether to execute a node. You can also reference a context variable from dialog node response conditions to show different responses depending on a value that is provided by an external service or by the user.

Learn more:

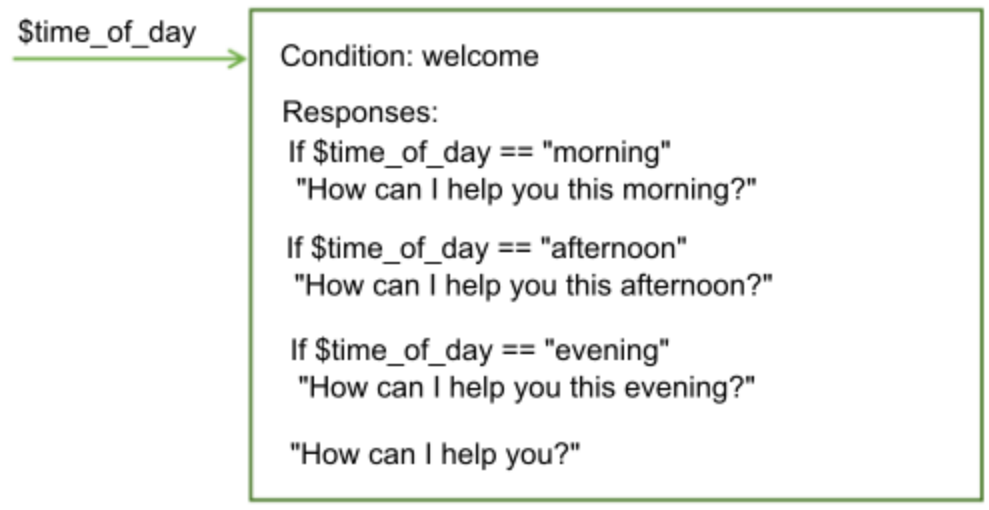
- Passing context from the application
- Passing context from node to node
- Defining a context variable

- Common context variable tasks
- Deleting a context variable
- Updating a context variable
- How context variables are processed
- Order of operation
- Adding context variables to a node with slots

Passing context from the application

Pass information from the application to the dialog by setting a context variable and passing the context variable to the dialog.

For example, your application can set a $time_of_day context variable, and pass it to the dialog that can use the information to tailor the greeting it shows to the user.



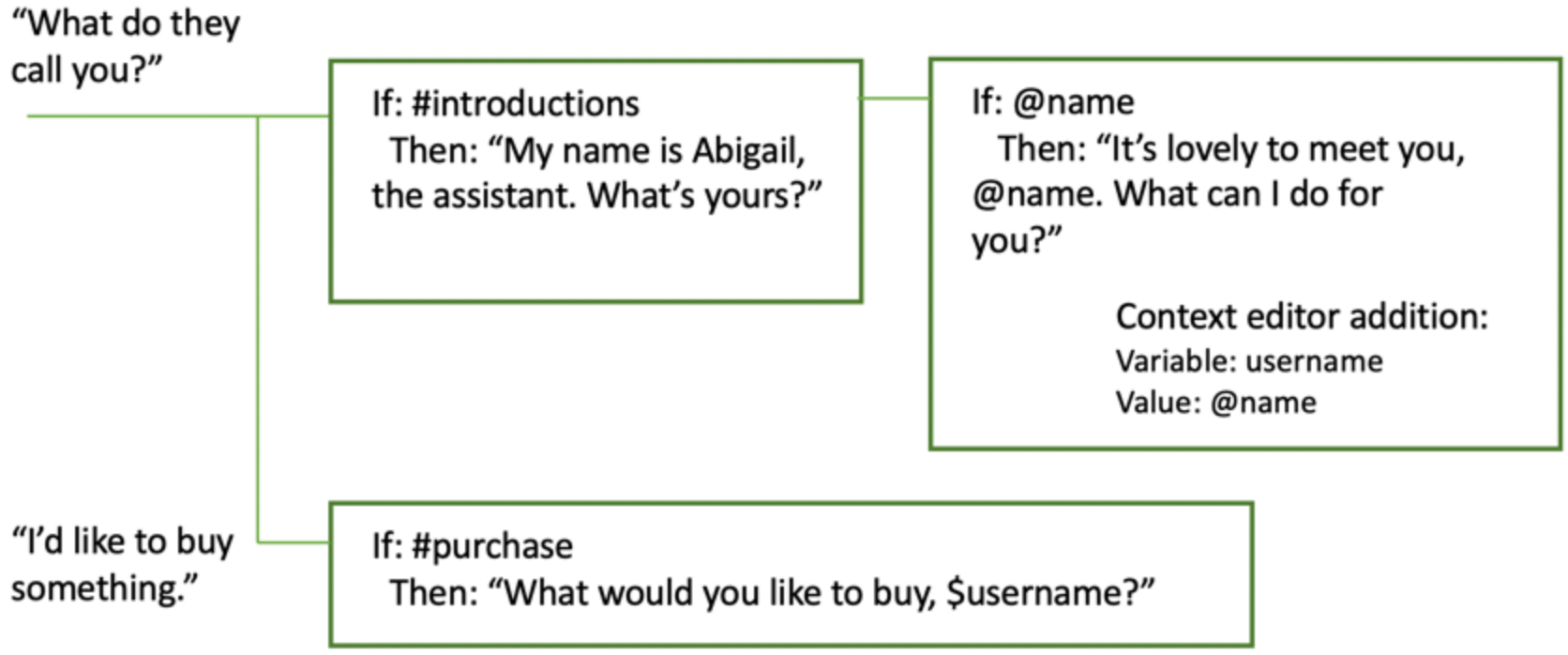
Time of day context

In this example, the dialog knows that the application sets the variable to one of these values: *morning*, *afternoon*, or *evening*. It can check for each value, and depending on which value is present, return the appropriate greeting. If the variable is not passed or has a value that does not match one of the expected values, then a more generic greeting is displayed to the user.

Passing context from node to node

The dialog can also add context variables to pass information from one node to another or to update the values of context variables. As the dialog asks for and gets information from the user, it can track the information and reference it later in the conversation.

For example, in one node you might ask users for their name, and in a later node address them by name.



Pass context from one node to another

In this example, the system entity @name is used to extract the user's name from the input if the user provides one. In the JSON editor, the username context variable is defined and set to the @name value. In a subsequent node, the $username context variable is included in the response to address the user by name.

Defining a context variable

Define a context variable by adding the variable name to the **Variable** field and adding a default value for it to the **Value** field in the node's edit view.

1. Click to open the dialog node to which you want to add a context variable.
2. Go to the *Assistant responds* section and click the menu icon ⋮.
3. Click **Open context editor**.
4. Add the variable name and value pair to the **Variable** and **Value** fields.
   - The `name` can contain any upper- and lowercase alphabet characters, numeric characters (0-9), and underscores.
   - The `value` can be any supported JSON type, such as a simple string variable, a number, a JSON array, or a JSON object.

The following table shows some examples of how to define name and value pairs for different types of values:

Variable	Value	Value Type
dessert	"cake"	String
age	18	Number
toppings_array	`["onions","olives"]`	JSON Array
full_name	`{"first":"John","last":"Doe"}`	JSON Object

Define name and value pairs

Then, to refer to these context variables, use the syntax `$name` where *name* is the name of the context variable that you defined.

For example, you might specify the following expression as the dialog response:

```
The customer, $age-year-old <? $full_name.first ?>, wants a pizza with <? $toppings_array.join(' and ') ?>, and then $dessert.
```

The resulting output is displayed as follows:

```
The customer, 18-year-old John, wants a pizza with onions and olives, and then cake.
```

You can use the JSON editor to define context variables also. You might prefer to use the JSON editor if you want to add a complex expression as the variable value. See Context variables in the JSON editor for more details.

Common context variable tasks

To store the entire string that was provided by the user as input, use `input.text`:

Variable	Value
repeat	`<?input.text?>`

Capturing user input

For example, the user input might be, `I want to order a device.` If the node response is, `You said: $repeat`, then the response would be displayed as, `You said: I want to order a device.`

To store the value of an entity in a context variable, use this syntax:

Variable	Value
place	`@place`

Capturing an entity mention

For example, the user input might be, `I want to go to Paris.` If your `@place` entity recognizes `Paris`, then your assistant saves `Paris` in the `$place` context variable.

To store the value of a string that you extract from the user's input, you can include a SpEL expression that uses the `extract` method to apply a regular expression to the user input. The following expression extracts a number from the user input, and saves it to the `$number` context variable.

Variable	Value
number	`<?input.text.extract('[\d]+',0)?>`

Using a String method

To store the value of a pattern entity, append .literal to the entity name. Using this syntax make sure that the exact span of text from user input that matched the specified pattern is stored in the variable.

Variable	Value
email	`<? @email.literal ?>`

Capturing a pattern entity value

For example, the user input is `Contact me at joe@example.com.` Your entity that is named `@email` recognizes the `name@domain.com` email format. By configuring the context variable to store `@email.literal`, you indicate that you want to store the part of the input that matched the pattern. If you omit the `.literal` property from the value expression, then the entity value name that you specified for the pattern is returned instead of the segment of user input that matched the pattern.

Deleting a context variable

To delete a context variable, set the variable to null.

Variable	Value
order_form	`null`

Nulling a context variable

Updating a context variable value

To update a context variable's value, define a context variable with the same name as the previous context variable, but this time, specify a different value for it.

When more than one node sets the value of the same context variable, the value for the context variable can change over the course of a conversation with a user. The value that is applied depends on which node is being triggered by the user in the course of the conversation. The value specified for the context variable in the last node that is processed overwrites any values that were set for the variable by nodes that were processed previously.

For information about how to update the value of a context variable when the value is a JSON object or JSON array data type, see Updating a context variable value in JSON

How context variables are processed

Where you define the context variable matters. The context variable is not created and set to the value that you specify for it until your assistant processes the part of the dialog node where you defined the context variable. In most cases, you define the context variable as part of the node response. When you do so, the context variable is created and given the specified value when your assistant returns the node response.

For a node with conditional responses, the context variable is created and set when the condition for a specific response is met and that response is processed. For example, if you define a context variable for conditional response #1 and your assistant processes response #2, then the variable that you defined for conditional response #1 is not set.

For information about adding context variables that you want your assistant to set as a user interacts with a node with slots, see Adding context variables to a node with slots.

Order of operation

When you define multiple variables to be processed together, the order in which you define them does not determine the order in which they are evaluated by your assistant. Your assistant evaluates the variables in random order. Don't set a value in the first context variable and expect to use it in the second variable because the first context variable might not be executed before the second one. For example, do not use two context variables to implement logic that checks whether the user input contains the word `Yes` in it.

Variable	Value
user_input	`<? input.text ?>`

contains_yes	<? $user_input.contains('Yes') ?>

Using two context variables to check for a value in user input

Instead, use a slightly more complex expression to avoid having to rely on the value of the first variable in your list (user_input) being evaluated before the second variable (contains_yes) is evaluated.

Variable	Value
contains_yes	<? input.text.contains('Yes') ?>

Using a single context variable

Adding context variables to a node with slots

For more information about slots, see Gathering information with slots.

To add a context variable that is processed after a response condition for a slot is met:

1. Open the node with slots in the edit view.
2. Click the **Customize slot** icon .
3. Click the **Options** icon , and then select **Enable condition**.
4. Click the **Customize handler** icon next to the response with which you want to associate the context variable.
5. Click the **Options** icon in the Assistant responds section, and then click **Open context editor**.
6. Add the variable name and value pair to the **Variable** and **Value** fields.

To add a context variable that is set or updated after a slot condition is met, complete the following steps:

1. Open the node with slots in the edit view.
2. Click the **Customize slot** icon .
3. Click the **Options** icon , and then select **Enable condition**.
4. Add the variable name and value pair in JSON format.

```
{
"time_of_day": "morning"
}
```

Note: You can't use the context editor to define context variables that are set during this phase of dialog node evaluation. You must use the JSON editor instead. For more information about using the JSON editor, see Context variables in the JSON editor.

Context variables in the JSON editor

You can also define a context variable in the JSON editor. Use the JSON editor if you are defining a complex context variable and want to be able to see the full SpEL expression as you add or change it.

The name and value pair must meet these requirements:

- The `name` can contain any upper- and lowercase alphabet characters, numeric characters (0-9), and underscores.

  Tip: You can include other characters, such as periods and hyphens, in the name. However, if you do, then you must specify the shorthand syntax `$(variable-name)` every time you reference the variable. See Expressions for accessing objects for more details.

- The `value` can be any supported JSON type, such as a simple string variable, a number, a JSON array, or a JSON object.

The following JSON sample defines values for the $dessert string, $toppings_array array, $age number, and $full_name object context variables:

```
{
  "context": {
    "dessert": "cake",
    "toppings_array": [
      "onions",
      "olives"
```

```
    ],
    "age": 18,
    "full_name": {
      "first": "Jane",
      "last": "Doe"
    }
  },
  "output":{}
}
```

To define a context variable in JSON format, complete the following steps:

1. Click to open the dialog node to which you want to add the context variable.

   > **Note:** Any existing context variable values that are defined for this node are displayed in a set of corresponding **Variable** and **Value** fields. If you do not want them to be displayed in the edit view of the node, you must close the context editor. You can close the editor from the same menu that is used to open the JSON editor; the following steps describe how to access the menu.

2. Click the **Options** icon ⋮ for the assistant response, and then click **Open JSON editor**.

   If the **Multiple conditioned responses** setting is enabled for the node, then you must first click the **Customize response** icon for the response with which you want to associate the context variable.

3. Add a `"context":{}` block if one is not present.

   ```
   {
     "context":{},
     "output":{}
   }
   ```

4. In the context block, add a `"name"` and `"value"` pair for each context variable that you want to define.

   ```
   {
     "context":{
       "name": "value"
   },
     "output": {}
   }
   ```

   In this example, a variable that is named `new_variable` is added to a context block that already contains a variable.

   ```
   {
     "context":{
       "existing_variable": "value",
       "new_variable":"value"
     }
   }
   ```

   To reference the context variable, use the syntax `$name` where *name* is the name of the context variable that you defined. For example, `$new_variable`.

Learn more:

- Deleting a context variable in JSON
- Updating a context variable value in JSON
- Setting one context variable equal to another

Deleting a context variable in JSON

To delete a context variable, set the variable to null.

```
{
  "context": {
    "order_form": null
  }
}
```

If you want to remove all trace of the context variable, you can use the JSONObject.remove(string) method to delete it from the context object. However, you must use a variable to perform the removal. Define the new variable in the message output so it isn't saved beyond the current call.

```
{
  "output": {
    "text" : {},
    "deleted_variable" : "<? context.remove('order_form') ?>"
  }
}
```

Alternatively you can delete the context variable in your application logic.

Updating a context variable value in JSON

In general, if a node sets the value of a context variable that is already set, then the previous value is overwritten by the new value.

Updating a complex JSON object

Previous values are overwritten for all JSON types except a JSON object. If the context variable is a complex type such as JSON object, a JSON merge procedure is used to update the variable. The merge operation adds any newly defined properties and overwrites any existing properties of the object.

In this example, a name context variable is defined as a complex object.

```
{
  "context": {
    "complex_object": {
      "user_firstname" : "Paul",
      "user_lastname" : "Pan",
      "has_card" : false
    }
  }
}
```

A dialog node updates the context variable JSON object with the following values:

```
{
  "complex_object": {
    "user_firstname": "Peter",
    "has_card": true
  }
}
```

The result is this context:

```
{
  "complex_object": {
    "user_firstname": "Peter",
    "user_lastname": "Pan",
    "has_card": true
  }
}
```

Updating arrays

If your dialog context data contains an array of values, you can update the array by appending values, removing a value, or replacing all the values.

Choose one of these actions to update the array. In each case, we see the array before the action, the action, and the array after the action is applied.

- **Append**: To add values to the end of an array, use the `append` method.

  For this dialog runtime context:

  ```
  {
    "context": {
      "toppings_array": ["onion", "olives"]
    }
  }
  ```

Make this update:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.append('ketchup', 'tomatoes') ?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ketchup", "tomatoes"]
  }
}
```

- **Remove**: To remove an element, use the `remove` method and specify its value or position in the array.
  - **Remove by value** removes an element from an array by its value.

    For this dialog runtime context:

    ```
    {
      "context": {
        "toppings_array": ["onion", "olives"]
      }
    }
    ```

    Make this update:

    ```
    {
      "context": {
        "toppings_array": "<? $toppings_array.removeValue('onion') ?>"
      }
    }
    ```

    Result:

    ```
    {
      "context": {
        "toppings_array": ["olives"]
      }
    }
    ```

  - **Remove by position**: Removing an element from an array by its index position:

    For this dialog runtime context:

    ```
    {
      "context": {
        "toppings_array": ["onion", "olives"]
      }
    }
    ```

    Make this update:

    ```
    {
      "context": {
        "toppings_array": "<? $toppings_array.remove(0) ?>"
      }
    }
    ```

    Result:

    ```
    {
      "context": {
        "toppings_array": ["olives"]
    ```

```
  }
}
```

- **Overwrite**: To overwrite the values in an array, set the array to the new values:

  For this dialog runtime context:

  ```
  {
    "context": {
      "toppings_array": ["onion", "olives"]
    }
  }
  ```

  Make this update:

  ```
  {
    "context": {
      "toppings_array": ["ketchup", "tomatoes"]
    }
  }
  ```

  Result:

  ```
  {
    "context": {
      "toppings_array": ["ketchup", "tomatoes"]
    }
  }
  ```

Setting one context variable equal to another

When you set one context variable equal to another context variable, you define a pointer from one to the other. Later, if the value of one of the variables changes, then the value of the other variable is changed also.

For example, if you specify a context variable as follows, then when the value of either `$var1` or `$var2` later changes, the value of the other changes too.

Variable	Value
var2	var1

Setting one context variable equal to another

Do not set one variable equal to another to capture a point in time value. With arrays, if you want to capture an array value that is stored in a context variable and use it later, create a new variable based on the current value of the variable instead.

For example, to create a copy of the values of an array at a certain point, add an array that is populated with the values for the existing array. To do so, you can use the following syntax:

```
{
"context": {
   "var2": "<? output.var2?:new JsonArray().append($var1) ?>"
 }
 }
```

Improving your conversation

Test your dialog, and organize your dialog nodes.

Testing your dialog

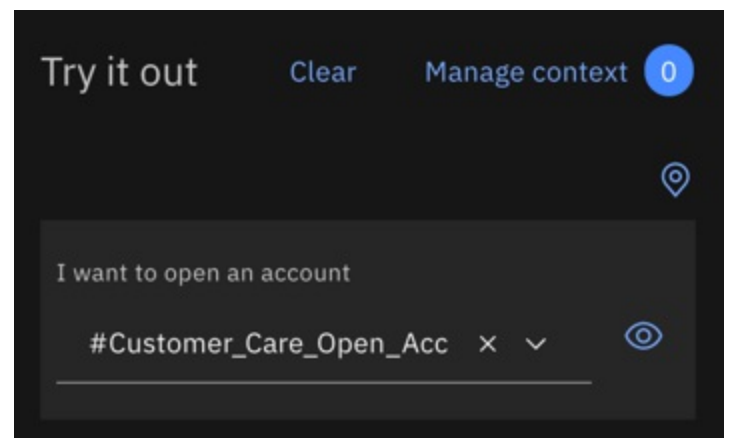
As you work on your dialog, you can test it at any time to see how the dialog responds to input.

Queries that you submit through the "Try it out" pane generate `/message` API calls, but they are not logged and do not incur charges.

1. From the Dialog page, click the **Try it** button.
2. In the chat pane, type some text and then press Enter.

3. Check the response to see if the dialog correctly interpreted your input and chose the appropriate response.

   The chat window indicates what intents and entities were recognized in the input.



Test dialog output

4. To see the top intents that were recognized, hover over the eye icon that is displayed next to the intent with the highest confidence score.
5. If the response is not what you expected it to be, you can take the following actions from the "Try it out" pane:
   - If you want to edit an entity that is recognized in the input, click the entity name to open it in the Entities page.
   - If the wrong intent is recognized, you can click the arrow next to the intent name to correct it or mark the topic as irrelevant. For more information, see Making training data improvements.
6. If you want to know which node in the dialog tree triggered a response, click the location icon next to it.

   The source node is given focus and the route that your assistant traversed through the tree to get to it is highlighted. It remains highlighted until you perform another action, such as entering a new test input.
7. To check or set the value of a context variable, click the **Manage context** link.

   Any context variables that you defined in the dialog are displayed.

   In addition, the following context variables are listed:

   - `$timezone`: The *Try it out* pane gets user locale information from the web browser and uses it to set the `$timezone` context variable. This context variable makes it easier to deal with time references in test dialog exchanges.
   - `$metadata`: This context variable contains a user ID name and value pair. Because no `user_id` is specified by default in the *Try it out* pane, the `conversation_id` is used as the `user_id` value. At run time, the ID value is typically passed to the assistant from whatever integration you are using. If you design a custom application, you set this value yourself.

   You can add a variable and set its value to see how the dialog responds in the next test dialog turn. This capability is helpful if, for example, the dialog is set up to show different responses based on a context variable value that is provided by the user.

   a. To add a context variable, specify the variable name, and press Enter.
   b. To define a default value for the context variable, find the context variable that you added in the list, and then specify a value for it.

   For more information, see Context variables.
8. Continue to interact with the dialog to see how the conversation flows through it.
   - To find and resubmit a test utterance, you can press the Up key to cycle through your recent inputs.
   - To remove prior test utterances from the chat pane and start over, click the **Clear** link. Not only are the test utterances and responses removed, but this action also clears the values of any context variables that were set as a result of your interactions with the dialog.

What to do next

Change the dialog to address the issues you see when testing:

- If you determine that the wrong intents or entities are being recognized, you might need to modify your intent or entity definitions.
- If the correct intents and entities are being recognized, but the wrong nodes are being triggered in your dialog, make sure that your conditions are written properly.

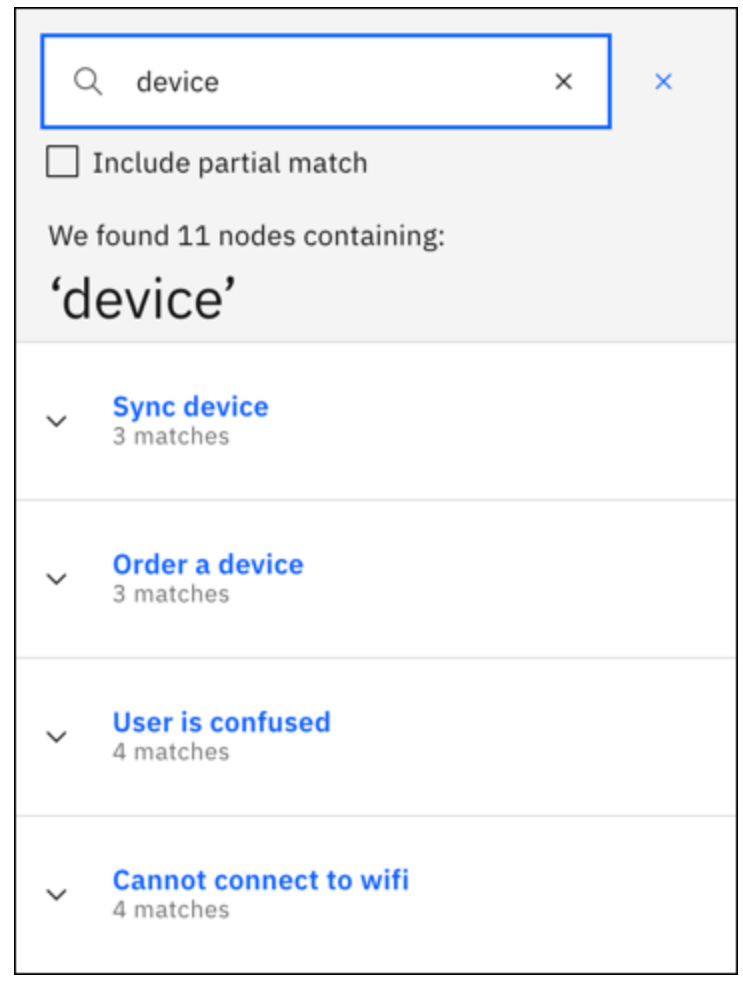
Searching your dialog

You can search the dialog to find one or more dialog nodes that mention a word or phrase.

1. Select the Search icon .
2. Enter a search term or phrase.

Note: The first time that you search, an index is created. You might be asked to wait while the text in your dialog nodes is indexed.

Nodes containing your search term, with corresponding examples, are shown. Select any result to open it for editing.



Search dialog

Copying a dialog node

You can duplicate a node to create an exact copy of it as a peer node directly after it in the dialog tree. The copied node itself is given the same name as the original node, but with `- copy n` appended to it, where *n* is a number that starts with 1. If you duplicate the same node more than once, then the *n* in the name increments by one for each copy to help you distinguish the copies from one another. If the node has no name, it is given the name `copy n`.

When you duplicate a node that has child nodes, the child nodes are duplicated also. The copied child nodes have the exact same names as the original child nodes. The only way to distinguish a copied child node from an original child node is the `copy` reference in the parent node name.

1. On the node you want to copy, click the **Node options** icon ⋮, and then select **Duplicate**.
2. Consider renaming the copied nodes or editing their conditions to make them distinct.

Moving a dialog node

Each node that you create can be moved elsewhere in the dialog tree.

You might want to move a previously created node to another area of the flow to change the conversation. You can move nodes to become siblings or peers in another branch.

1. On the node you want to move, click the **Node options** icon ⋮, and then select **Move**.
2. Select a target node that is located in the tree near where you want to move this node. Choose whether to place this node before or after the target node, or to make it a child of the target node.

Organizing the dialog with folders

You can group dialog nodes together by adding them to a folder. Reasons reasons to group nodes include:

- To keep nodes that address a similar subject together to make them easier to find. For example, you might group nodes that address questions about user accounts in a *User account* folder and nodes that handle payment-related queries in a *Payment* folder.
- To group a set of nodes that you want the dialog to process only if a certain condition is met. Use a condition, such as `$isPlatinumMember`, to group nodes that offer extra services that are processed only if the current user is entitled to receive the extra services.
- To hide nodes from the runtime while you work on them. You can add the nodes to a folder with a `false` condition to prevent them from being processed.

These characteristics of the folder impact how the nodes in a folder are processed:

- Condition: If no condition is specified, then your assistant processes the nodes within the folder directly. If a condition is specified, your assistant first

evaluates the folder condition to determine whether to process the nodes within it.

- Customizations: Any configuration settings that you apply to the folder are inherited by the nodes in the folder. If you change the digression settings of the folder, for example, the changes are inherited by all the nodes in the folder.
- Tree hierarchy: Nodes in a folder are treated as root or child nodes based on whether the folder is added to the dialog tree at the root or child level. Any root level nodes that you add to a root level folder continue to function as root nodes; they do not become child nodes of the folder, for example. However, if you move a root level node into a folder that is a child of another node, then the root node becomes a child of that other node.

Folders have no impact on the order in which nodes are evaluated. Nodes continue to be processed from first to last. As your assistant follows the tree, if the folder has no condition or its condition is true, it immediately processes the first node in the folder, and continues down the tree in order. If a folder does not have a folder condition, then each node in the folder is treated like any other individual node in the tree.

Adding a folder

To add a folder to a dialog tree, complete the following steps:

1. From the tree view of dialog, click **Add folder**.

   The folder is added to the end of the dialog tree, just before the **Anything else** node. Unless an existing node in the tree is selected, in which case, it is added after the selected node.

   If you want to add the folder elsewhere in the tree, from the node before the spot where you want to add it, click the **Node options** icon ⋮, and then select **Add folder**.

   You can add a folder after a child node within an existing dialog branch. To do so, click the **Node options** icon ⋮ on the child node, and then select **Add folder**.

   The folder is opened in edit view.

2. **Optional**: Name the folder.
3. **Optional**: Define a condition for the folder.

   If you do not specify a condition, `true` is used, meaning the nodes in the folder are always processed.

4. Add dialog nodes to the folder.
   - To add existing dialog nodes to the folder, you must move them to the folder one at a time.

     On the node that you want to move, click the **Node options** icon ⋮, select **Move**, and then click the folder. Select **To folder** as the move-to target.

     > **Tip:** As you move nodes, they are added at the start of the tree within the folder. Therefore, if you want to retain the order of a set of consecutive root dialog nodes, for example, move them starting with the last node first.

   - To add a dialog node to the folder, click the **Node options** icon ⋮ on the folder, and then select **Add node to folder**.

     The dialog node is added to the end of the dialog tree within the folder.

Deleting a folder

You can delete either a folder alone or the folder and all of the dialog nodes in it.

To delete a folder, complete the following steps:

1. From the tree view of the **Dialog** tab, find the folder that you want to delete.
2. Click the **Node options** icon ⋮ on the folder, and then select **Delete**.
3. Do one of the following things:
   - To delete the folder only, and keep the dialog nodes that are in the folder, deselect the **Delete the nodes inside the folder** checkbox, and then click **Yes, delete it**.
   - To delete the folder and all of the dialog nodes in it, click **Yes, delete it**.

If you deleted the folder only, then the nodes that were in the folder are displayed in the dialog tree in the spot where the folder used to be.

Dialog node limits

The number of dialog nodes you can create per skill depends on your plan type.

Plan	Dialog nodes per skill
Enterprise	100,000
Premium (legacy)	100,000
Plus	100,000
Trial	25,000
Lite	25,000

Plan details

The `welcome` and `anything_else` dialog nodes that are prepopulated in the tree do count toward the total.

Tree depth limit: The dialog supports 2,000 dialog node descendants; the dialog works best with 20 or fewer.

Finding a dialog node by its node ID

You can search for a dialog node by its node ID. Enter the full node ID into the search field. You might want to find the dialog node that is associated with a known node ID for any of the following reasons:

- You are reviewing logs, and the log refers to a section of the dialog by its node ID.
- You want to map the node IDs listed in the `nodes_visited` property of the API message output to nodes that you can see in your dialog tree.
- A dialog runtime error message informs you about a syntax error, and uses a node ID to identify the node you need to fix.

Another way to discover a node based on its node ID is by following these steps:

1. From the Dialog page, select any node in your dialog tree.
2. Close the edit view if it is open for the current node.
3. In your web browser's location field, a URL should display that has syntax similar to the following example:

   ```
   https://{location}.assistant.watson.cloud.ibm.com/{location}/{instance-id}/skills/{skill-id}/build/dialog#node={node-id}
   ```

4. Edit the URL by replacing the current `{node-id}` value with the ID of the node that you want to find, and then submit the new URL.
5. If necessary, highlight the edited URL again, and resubmit it.

The page refreshes, and shifts focus to the dialog node with the node ID that you specified. If the node ID is for a slot, a Found or Not found slot condition, a slot handler, or a conditional response, then the node in which the slot or conditional response is defined gets focus and the corresponding modal is displayed.

Tip: If you still cannot find the node, you can export the dialog skill and use a JSON editor to search the skill JSON file.

How many nodes are in my dialog?

To see the number of dialog nodes in a dialog skill, do one of the following things:

- If it is not associated with an assistant already, add the dialog skill to an assistant, and then view the skill tile from the main page of the assistant. The *trained data* section lists the number of dialog nodes.
- Send a GET request to the `/dialog_nodes` API endpoint, and include the `include_count=true` parameter. For example:

  ```
  $ curl -u "apikey:{apikey}" "{url}/v1/workspaces/{workspace_id}/dialog_nodes?version=2018-09-20&include_count=true"
  ```

  where {url} is the appropriate URL for your instance. For more information, see Service endpoint.

  In the response, the `total` attribute in the `pagination` object contains the number of dialog nodes.

If the total seems larger than you expected, it might be because the dialog that you build from the application is converted into a JSON object. Some fields that appear to be part of a single node are structured as separate dialog nodes in the underlying JSON object.

- Each node and folder is represented as its own node.
- Each conditional response that is associated with a single dialog node is represented as an individual node.

- For a node with slots, each slot, slot found response, slot not found response, slot handler, and if set, the "prompt for everything" response is an individual node. In effect, one node with three slots might be equivalent to eleven dialog nodes.

Controlling the conversational flow

Understand how your dialog is processed when a person interacts with your assistant at run time. Learn to use the conversation flow to your advantage and alter the flow if necessary.

You can impact the flow of a conversation with the following features:

- Disambiguation
- Digressions

Disambiguation

Disambiguation instructs your assistant to ask the customer for help when more than one dialog node can respond to a customer's input. Instead of guessing which node to process, your assistant shares a list of the top node options with the user, and asks the user to pick the right one.

Assistant asks for clarification

Disambiguation is triggered when the following conditions are met:

- The confidence scores of the runner-up intents that are detected in the user input are close in value to the confidence score of the top intent.
- The confidence score of the top intent is above 0.2.

Even when these conditions are met, disambiguation does not occur unless two or more independent nodes in your dialog meet the following criteria:

- The node condition includes one of the intents that triggered disambiguation.
- A description of the node's purpose is provided for the node in the node name field. (Alternatively, a description can be included in the external node name field.)

A node with a Boolean node condition that evaluates to true is likely to be included in the disambiguation list. For example, if the node checks for an entity type, and the entity is mentioned in the user input, it is eligible to be included in the list. Such nodes do not trigger disambiguation, but if disambiguation is triggered, they are likely to be included in the resulting disambiguation list.

Learn more

- Disambiguation example
- Editing the disambiguation configuration
- Choosing nodes to disable
- Disabling disambiguation
- Handling none of the above
- Testing disambiguation

Disambiguation example

For example, you have a dialog that has two nodes with intent conditions that address cancellation requests. The conditions are:

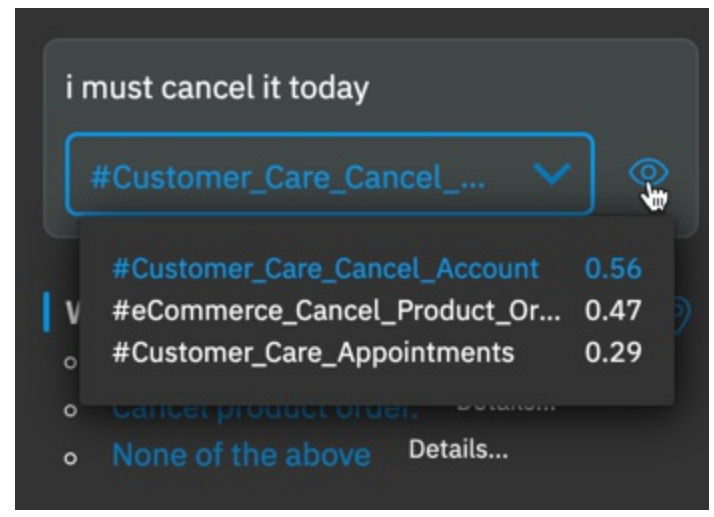
- eCommerce_Cancel_Product_Order
- Customer_Care_Cancel_Account

If the user says `i must cancel it today`, then the following intents might be detected in the input:

```
$ [
```

{"intent":"Customer_Care_Cancel_Account","confidence":0.5602024316787719},
{"intent":"eCommerce_Cancel_Product_Order","confidence":0.46903514862060547},
{"intent":"Customer_Care_Appointments","confidence":0.29033891558647157},
{"intent":"General_Greetings","confidence":0.2894785046577454},

If you test from the "Try it out" pane, you can hover over the eye icon to see the top three intents that were recognized in the test input.

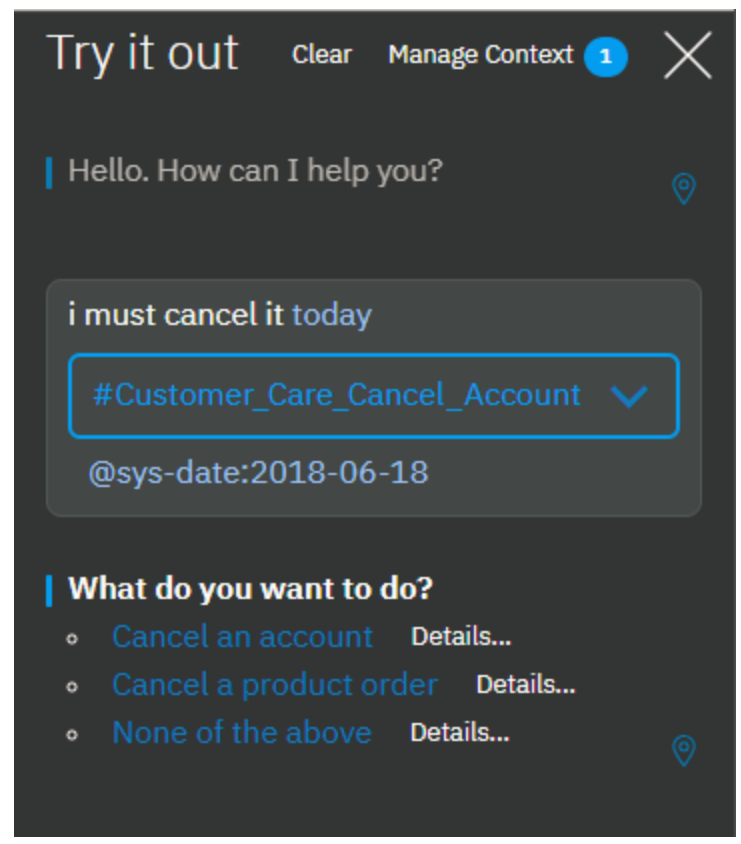


Top intents

Your assistant is `0.5618281841278076` (56%) confident that the user goal matches the `#Customer_Care_Cancel_Account` intent. If another intent has a confidence score that is close to the score of this top intent, then disambiguation is triggered. In this example, the `#eCommerce_Cancel_Product_Order` intent has a close confidence score of 46%.

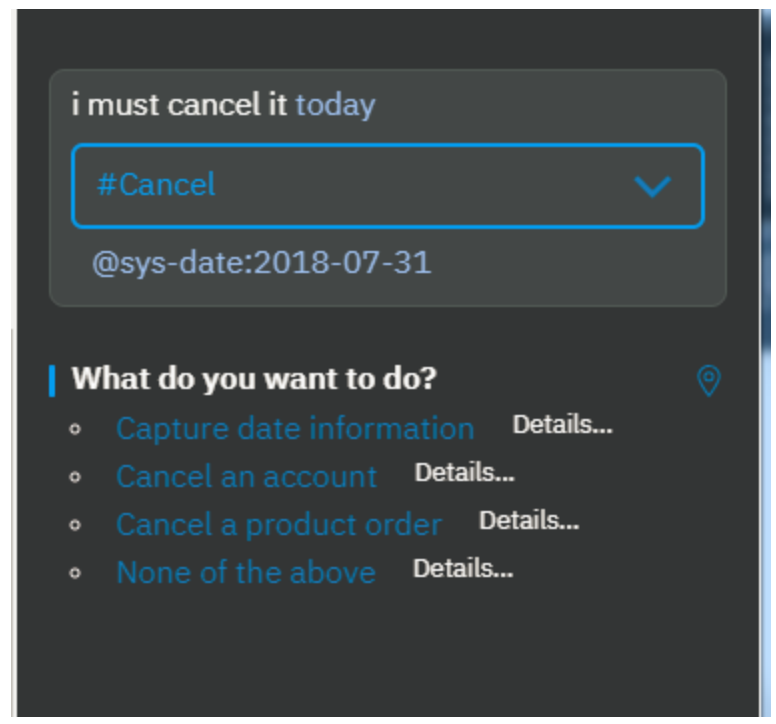
As a result, when the user says `i must cancel it today`, both dialog nodes are likely to be considered viable candidates to respond. To determine which dialog node to process, the assistant asks the user to pick one. And to help the user choose between them, the assistant provides a short summary of what each node does. The summary text is extracted directly from the node's *name* field. If present and if a description is added to it, then the text is taken from the *external node name* field instead.

Note: The description that is displayed in the disambiguation list comes from the name (or external node name) of the last node that is processed in the branch where the intent match occurs. For more information, see Disable jumped-to utility nodes.



Choose options

Notice that your assistant recognizes the term `today` in the user input as a date, a mention of the `@sys-date` entity. If your dialog tree contains a node that conditions on the `@sys-date` entity, then it is also likely to be included in the list of disambiguation choices. This image shows it included in the list as the *Capture date information* option.



Capture date information

Editing the disambiguation configuration

Disambiguation is enabled automatically for all new dialog skills. You can change the settings that are applied automatically to disambiguation from the *Options* page.

To edit the disambiguation settings, complete the following steps:

1. In **Options**, click **Disambiguation**.
2. In the **Disambiguation message** field, add text to show before the list of dialog node options. For example, *What do you want to do?*
3. In the **Anything else** field, add text to display as an extra option that users can pick if none of the other dialog node options reflect what the user wants to do. For example, *None of the above*.

   Keep the message short, so it displays inline with the other options. The message must be fewer than 512 characters. For information about what your assistant does if a user chooses this option, see Handling none of the above.
4. If you want to limit the number of disambiguation options that can be displayed to a user, then in the **Maximum number of suggestions** field, specify a number between 2 and 5.

Your changes are automatically saved.

Tip: You can use the API to modify more disambiguation settings. These settings include the disambiguation sensitivity, which affects how often disambiguation is triggered and how many choices are included. For more information, see the API Reference.

Choosing nodes to not show as disambiguation options

All nodes are eligible to be included in the disambiguation list.

- Nodes at any level of the tree hierarchy are included.
- Nodes that condition on intents, entities, special conditions, context variables, or any combination of these values are included.

Consider hiding some nodes from the disambiguation list.

- **Hide root nodes with `welcome` and `anything_else` conditions**

  Unless you added extra functions to these nodes, they are typically not useful options to include in a disambiguation list.

- **Hide jumped-to utility nodes**

  Important: The text in the disambiguation list is populated from the node name (or external node name) of the *last node that is processed* in the branch where the node condition is matched.

  Don't include utility nodes, such as thanking the user, saying goodbye, or asking for answer quality feedback, in the disambiguation list instead of a phrase that describes the purpose of the matched root node.

  For example, maybe a root node with the matching intent condition of `#store-location` jumps to a node that asks users if they are satisfied with the response. If the `#check_satisfaction` node has a node name and disambiguation, then the name for the jumped-to node is displayed in the disambiguation list. As a result, `Check satisfaction` is displayed in the disambiguation list to represent the `#store-location` branch instead of the `Get`

`store location` name from the root node.

- **Hide root nodes that condition on an entity or context variable only**

  Only a node with a matched intent can trigger disambiguation. However, when triggered, any node with a condition that matches is included in the disambiguation list. When your dialog includes nodes that condition on entities, for example, the order of nodes in the tree hierarchy can become significant in unexpected ways.

  - The order of nodes impacts whether disambiguation is triggered at all. Look at the scenario that is used earlier to introduce disambiguation, for example. If the node that conditions on `@sys-date` was placed higher in the dialog tree than the nodes that condition on the `#Customer_Care_Cancel_Account` and `#eCommerce_Cancel_Product_Order` intents, disambiguation would never be triggered when a user enters `i must cancel it today`. Your assistant would consider the date that is mentioned (`today`) to be more important than the intent references due to the placement of the corresponding nodes in the tree.

  - The order of nodes impacts which nodes are included in the disambiguation options list. Sometimes a node is not listed as a disambiguation option as expected. This can happen if a condition value is also referenced by a node that is *not* eligible for inclusion in the disambiguation list for some reason. For example, an entity might trigger a node that is situated earlier in the dialog tree but is not enabled for disambiguation. If the same entity is the only condition for a node that *is* enabled for disambiguation, but is lower in the tree, then it might not be a disambiguation option because your assistant never reaches it. If it matched against the earlier node and was omitted, your assistant might not process the later node.

Hiding nodes from disambiguation altogether

You can prevent every node in a dialog or an individual dialog node from being included in the disambiguation list.

To disable disambiguation entirely:

1. In **Options**, click **Disambiguation**.
2. Set the switch to **Off**.

To prevent a single dialog node from being included in the disambiguation list:

1. Click a dialog node to open it in the edit view.
2. Click **Settings**.
3. Set the **Show node name** switch to **Off**.

   Plus If you added a node summary description to the **external node name** field instead of the *name* field, remove it.

   The *external node name* field serves two purposes. It provides information about the node to customers when it is included in a disambiguation list. It also describes the node in a chat summary that is shared with service desk agents when a conversation is transferred to a person. The *external node name* field is only visible in skills that are part of a paid plan instance. If the *external node name* field contains text, its text is used, whether or not there is text in the *name* field.

For each node, test scenarios in which you expect the node to be included in the disambiguation options list. Testing gives you a chance to adjust the node order or other factors that might impact how well disambiguation works at run time. See Testing disambiguation.

Prioritizing a node over disambiguation

Some nodes are important enough to your customer that you want them to be returned on their own, outside of a disambiguation list, when the assistant is confident that the node meets a customer need.

For example, you might have a node that matches the `#stolen_card` intent. Whenever an incoming message suggests that a customer wants to report a stolen credit card, you don't want your assistant's response to be lost in a disambiguation list of options. You want the assistant to respond with a single answer that helps the customer address this urgent matter.

To design your dialog to prioritize a single node over disambiguation, complete the following steps:

1. In the node that conditions on the intent, click **Customize** to enable multiple conditioned responses.
2. Add a conditioned response with the following condition:

   `intent.confidence > n`

   where `n` is a confidence score that makes sense for your training data. For example:

   `intent.confidence > 0.7`
3. Move the response up to be first in the list of conditioned responses.

4. Click the gear icon to customize the conditioned response.
5. In the Assistant responds section, open the context editor.
6. Add the following context variable:

Variable	Value
`system`	`{"prevent_disambiguation":true}`

Context variable

7. Click **Save**.

   Alternatively, you can add a root-level node with a condition such as:

   `#stolen_card && intent.confidence > 0.7`

   Place this node higher in the tree than the node that conditions on `#stolen_card`, which allows the node to be included in a disambiguation list.

8. Test your dialog. Make sure that the node's response is returned instead of a disambiguation list when the appropriate confidence score threshold is met.

Handling none of the above

When a user clicks the *None of the above* option, your assistant strips the intents that were recognized in the user input from the message and submits it again. This action typically triggers the anything else node in your dialog tree.

Tip: To customize the response that is returned in this situation, you can add a root node with a condition that checks for a user input with no recognized intents (the intents are stripped, remember) and contains a `suggestion_id` property. A `suggestion_id` property is added by your assistant when disambiguation is triggered.

Add a root node with the following condition:

```
intents.size()==0 && input.suggestion_id
```

This condition is met only by input that triggers a set of disambiguation options of which the user says none match the goal.

Add a response that tells users that know that you understand that none of the options that were suggested met their needs, and take appropriate action.

Again, the placement of nodes in the tree matters. If a node that conditions on an entity type that is mentioned in the user input is higher in the tree than this node, its response is displayed instead.

Testing disambiguation

Important: When testing, the order in which the options are listed might change. In fact, the options themselves that are included in the disambiguation list might change from one test to the next.

This behavior is intended. As part of development that is in progress to help the assistant learn automatically from user choices, the choices and their order in the disambiguation list is being randomized on purpose. Changing the order helps to avoid the bias that can be introduced by a percentage of people who always pick the first option without carefully reviewing all of their choices beforehand.

To test disambiguation, complete the following steps:

1. From the "Try it out" pane, enter a test utterance that is a good candidate for disambiguation, meaning two or more of your dialog nodes are configured to address utterances like it.
2. If the response doesn't contain a list of dialog node options for you to choose from, check that you added summary information to the node name (or external node name) fields.
3. If disambiguation is still not triggered, it might be that the confidence scores for the nodes are not as close in value as you thought.
   - To see the confidence scores of the top three intents that were detected in the input, hover over the eye icon in the "Try it out" pane.
   - To see the confidence scores of all the intents that are detected in the user input, temporarily add `<? intents ?>` to the end of the node response for a node that you know will be triggered.

     This SpEL expression shows the intents that were detected in the user input as an array. The array includes the intent name and the level of

confidence that your assistant has that the intent reflects the user's intended goal.

- To see which entities, if any, were detected in the user input, you can temporarily replace the current response with a single text response that contains the SpEL expression `<? entities ?>`.

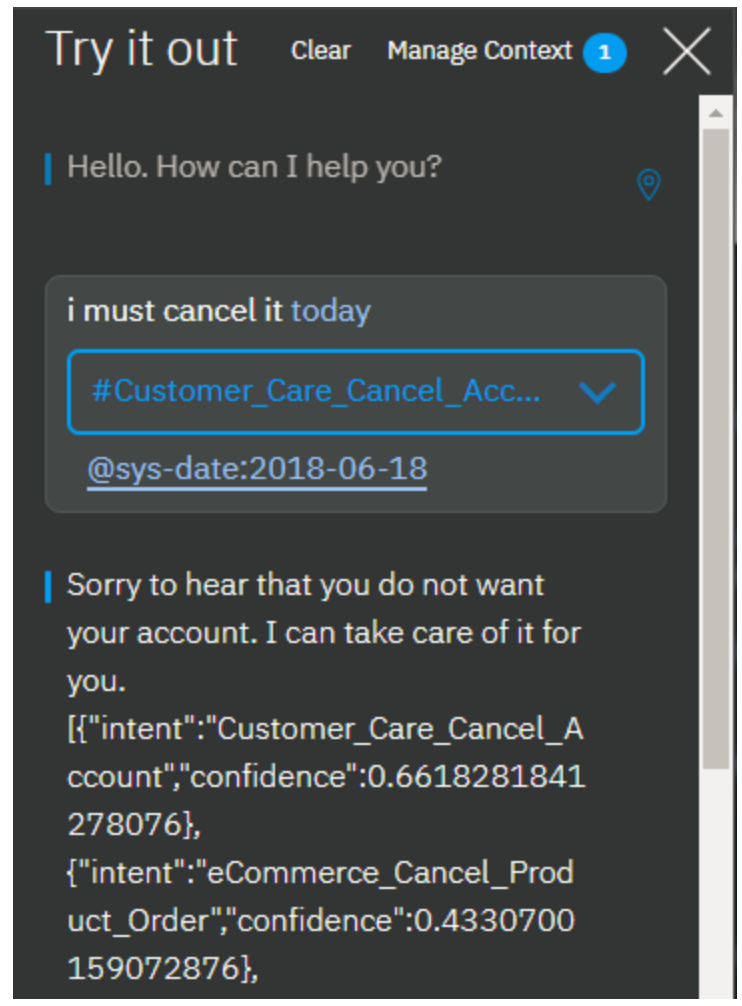
  This SpEL expression shows the entities that were detected in the user input as an array. The array includes the entity name, location of the entity mention, the entity mention string, and the level of confidence that the term is a mention of the entity type specified.

- To see details for all of the artifacts, including other properties such as context variable value at the time of the call, inspect the entire API response. See Viewing API call details.

4. Temporarily remove the description in the *name* field (or *external node name* field) for at least one of the nodes that you anticipate as a disambiguation option.

5. Enter the test utterance into the "Try it out" pane again.

   If you added the `<? intents ?>` expression to the response, then the text includes a list of the intents that your assistant recognized in the test utterance, and includes the confidence score for each one.



Intents and confidence scores

After you finish testing, remove any SpEL expressions that you appended to node responses, or add back any original responses that you replaced with expressions, and repopulate any *name* or *external node name* fields from which you removed text.

Digressions

A digression occurs when a user is in a dialog flow that addresses one goal, but switches topics to initiate a dialog flow that addresses a different goal. If none of the nodes in the dialog branch match the goal of the user's last input, the conversation checks the root node conditions for an appropriate match. The digression settings that are available per node give you the ability to tailor this behavior even more.

With digression settings, you can allow the conversation to return to the dialog flow that was interrupted when the digression occurred. For example, the user might be ordering a new phone, but switches topics to ask about tablets. Your dialog can answer the question about tablets, and then bring the user back to where they left off with the phone order. Allowing digressions to occur and return gives your users more control over the flow of the conversation at run time. They can change topics, follow a dialog flow about the unrelated topic to its end, and then return to where they were before. The result is a dialog flow that more closely simulates a human-to-human conversation.

Before you begin

As you test your overall dialog, decide when and where it makes sense to allow digressions and returns from digressions to occur. The following digression controls are applied to the nodes automatically.

- Every root node in your dialog is configured to allow digressions to target them by default. Child nodes cannot be the target of a digression.
- Nodes with slots are configured to prevent digressions away. All other nodes are configured to allow digressions away. However, the conversation cannot digress away from a node under the following circumstances:

- If any of the child nodes of the current node contain the `anything_else` or `true` condition. These conditions are special in that they always evaluate to be true. Because of their known behavior, they are often used in dialogs to force a parent node to evaluate a specific child node in succession. To prevent breaking existing dialog flow logic, digressions are not allowed in this case. Before you can enable digressions away from such a node, you must change the child node's condition to something else.
- If the node is configured to jump to another node or skip user input after it is processed. The final step section of a node specifies what happens after the node is processed. When the dialog is configured to jump directly to another node, it is often to make sure that a specific sequence is followed. And when the node is configured to skip user input, it is equivalent to forcing the dialog to process the first child node after the current node in succession. To prevent breaking existing dialog flow logic, digressions are not allowed in either of these cases. Before you can enable digressions away from this node, you must change what is specified in the final step section.

Customizing digressions

You do not define the start and end of a digression. The user is entirely in control of the digression flow at run time. You can specify how each node is included in a user-led digression. For each node, you can configure whether a digression:

- Starts from and leaves the node
- That starts elsewhere can target and enter the node
- That starts elsewhere and enters the node must return to the interrupted dialog flow after the current dialog flow is completed

To change the digression behavior for an individual node, complete the following steps:

1. Click the node to open its edit view.
2. Click **Customize**, and then click the **Digressions** tab.

   The configuration options differ depending on whether the node you are editing is a root node, a child node, a node with children, or a node with slots.

   Digressions away from this node

   If the circumstances listed earlier do not apply, then you can make the following choices:

   - **All node types**: Choose whether to allow users to digress away from the current node before they reach the end of the current dialog branch.
   - **All nodes that have children**: Choose if you want the conversation to return to the current node if the node's response has been displayed and its child nodes are incidental to the goal. Set the **Allow return from digressions triggered after this node's response** switch to **No** to prevent the dialog from returning to the current node and continuing to process its branch.

     For example, if the user asks, `Do you sell cupcakes?` and the response `We offer cupcakes in a variety of flavors and sizes` is displayed before the user changes subjects, you might not want the dialog to return to where it left off. Especially, if the child nodes address possible follow-up questions from the user and can safely be ignored.

     However, if the node relies on its child nodes to address the question, then you might want to force the conversation to return and continue processing the nodes in the current branch. For example, the initial response might be, `We offer cupcakes in all shapes and sizes. Which menu do you want to see: gluten-free, dairy-free, or regular?`. If the user changes subjects, you might want the dialog to return so the user can pick a menu type and get the information that they wanted.
   - **Nodes with slots**: Choose whether you want to allow users to digress away from the node before all of the slots are filled. Set the **Allow digressions away while slot filling** switch to **Yes** to enable digressions away.

     If enabled, when the conversation returns from the digression, the prompt for the next unfilled slot is displayed to encourage the user to continue providing information. If disabled, then any inputs that the user submits which do not contain a value that can fill a slot are ignored. However, you can address unsolicited questions that you anticipate your users might ask while they interact with the node by defining slot handlers. For more information, see Adding slots.
   - **Nodes with slots**: Choose whether the user is allowed to digress away if they return to the current node by selecting the **Only digress from slots to nodes that allow returns** checkbox.

     When selected, as the dialog looks for a node to answer the user's unrelated question, it ignores any root nodes that are not configured to return after the digression. Select this checkbox if you want to prevent users from being able to permanently leave the node before they finish filling the required slots.

   Digressions into this node

   You can make the following choices about how digressions into a node behave:

   - Prevent users from being able to digress into the node. See Disabling digressions into a root node for more details.
   - When digressions into the node are enabled, choose whether the dialog must go back to the dialog flow that it digressed away from. After the

current node's branch is done being processed, the dialog flow goes back to the interrupted node. To make the dialog return afterward, select **Return after digression**.

3. Click **Apply**.
4. Use the "Try it out" pane to test the digression behavior.

   Again, you cannot define the start and end of a digression. The user controls where and when digressions happen. You can apply settings that determine how a single node participates in one. Because digressions are so unpredictable, it is hard to know how your configuration decisions impact the overall conversation. To truly see the impact of the choices you made, you must test the dialog.

The #reservation and #cuisine nodes represent two dialog branches that can participate in a single user-directed digression. The digression settings that are configured for each individual node are what make this type of digression possible at run time.

Digression usage tips

Custom return message: For any nodes where you enable returns from digressions away, consider adding wording that tells users where they are returning to when they left off in a previous dialog flow. In your text response, use a special syntax to add two versions of the response.

If you don't add a message, the same text response is displayed a second time to tell users that they returned to the node they digressed away from. You can make it clearer to users that they returned to the original conversation thread by specifying a unique message to be displayed when they return.

For example, if the original text response for the node is, `What's the order number?`, then you might want to display a message like, `Now let's get back to where we left off. What is the order number?` when users return to the node.

To do so, use the following syntax to specify the node text response:

```
<? (returning_from_digression)? "post-digression message" : "first-time message" ?>
```

For example:

```
<? (returning_from_digression)? "Now, let's get back to where we left off.
What is the order number?" : "What's the order number?" ?>
```

> **Note:** You cannot include SpEL expressions or shorthand syntax in the text responses that you add. In fact, you cannot use shorthand syntax at all. Instead, you must build the message by concatenating the text strings and full SpEL expression syntax together to form the full response.

For example, use the following syntax to include a context variable in a text response that you would normally specify as `What can I do for you, $username?`:

```
$ <? (returning_from_digression)? "Where were we, " +
context["username"] + "? Oh right, I was asking what can I do
for you today." : "What can I do for you today, " +
context["username"] + "?" ?>
```

For full SpEL expression syntax details, see Expression for accessing objects.

Preventing returns: In some cases, you might want to prevent a return to the interrupted conversation flow based on a choice the user makes in the current dialog flow. You can use special syntax to prevent a return from a specific node.

For example, you might have a node that conditions on `#General_Connect_To_Agent` or a similar intent. When triggered, if you want to get the user's confirmation before you transfer them to an external service, you might add a response such as `Do you want me to transfer you to an agent now?`. You might add two child nodes that condition on `#yes` and `#no`.

The best way to manage digressions for this type of branch is to set the root node to allow digression returns. However, on the `#yes` node, include the SpEL expression `<? clearDialogStack() ?>` in the response. For example:

```
OK. I will transfer you now. <? clearDialogStack() ?>
```

This SpEL expression prevents the digression return from happening from this node. When a confirmation is requested, if the user says yes, the proper response is displayed, and the dialog flow that was interrupted is not resumed. If the user says no, then the user is returned to the flow that was interrupted.

Disabling digressions into a root node

When a flow digresses into a root node, it follows the course of the dialog that is configured for that node. The flow might process a series of child nodes before the end of the node branch, and then go back to the dialog flow that was interrupted. By testing, you might find that a root node is triggered too often, at unexpected times, or the dialog is too complex and sends the user too far off course. If you determine that you would rather not allow users to

digress into it, you can configure the root node to not allow digressions in.

To disable digressions into a root node altogether, complete the following steps:

1. Click to open the root node that you want to edit.
2. Click **Customize**, and then click the **Digressions** tab.
3. Set the **Allow digressions into this node** to **Off**.
4. Click **Apply**.

If you decide that you want to prevent digressions into several root nodes, but do not want to edit each one individually, you can add the nodes to a folder. From the *Customize* page of the folder, you can set the **Allow digressions into this node** switch to **Off** to apply the configuration to all of the nodes. For more information, see Organizing the dialog with folders.

Design considerations

Avoid fallback node proliferation: Many dialog designers include a node with a `true` or `anything_else` condition at the end of every dialog branch as a way to prevent users from getting stuck in the branch. This design returns a generic message if the user input does not match anything that you anticipated and included a specific dialog node to address. However, users cannot digress away from dialog flows that use this approach.

Evaluate any branches that use this approach to determine whether it would be better to allow digressions away from the branch. If the user's input does not match anything that you anticipated, it might find a match against an entirely different dialog flow in your tree. Rather than responding with a generic message, you can effectively put the rest of the dialog to work to try to address the user's input. And the root-level `Anything else` node can always respond to input that none of the other root nodes can address.

Reconsider jumps to a closing node: Many dialogs are designed to ask a standard closing question, such as, `Did I answer your question today?` Users cannot digress away from nodes that are configured to jump to another node. So, if you configure all of your final branch nodes to jump to a common closing node, digressions cannot occur. Consider tracking user satisfaction through metrics or some other means.

Test possible digression chains : If a user digresses from the current node to another node that allows digressions, the user might digress away from that other node, and repeat this pattern. If the starting node in the digression chain is configured to return after the digression, then the user will eventually be brought back to the current dialog node. In fact, any subsequent nodes in the chain that are configured not to return are excluded from being considered as digression targets. Test scenarios that digress multiple times to determine whether individual nodes function as expected.

Step conditions are not reevaluated on digression returns : Because the step condition is already evaluated before the digression, it is not evaluated again upon the return of the digression. If your conversational flow requires the step condition to be evaluated again, then add reprompting at the same step where the digression started. If you do not want to reprompt at the same step and to prevent digression returns, do the following steps:

1. Go to **Home** > **Actions** > **Created by you**.
2. Select the action in which you want to prevent digression returns.
3. Click **Action settings**.
4. Switch the toggle to **On** in **Change conversation topic**.
5. Select **Never return to original action after completing this action** .
6. Click **Save**.

The current node gets priority: Nodes outside the current flow are only considered as digression targets if the current flow cannot address the user input. In a node with slots that allows digressions away, make it clear to users what information is needed, and to add confirmation statements that are displayed after the user provides a value.

Any slot can be filled during the slot-filling process. So, a slot might capture user input unexpectedly. For example, you might have a node with slots that collects the information necessary to make a dinner reservation. One of the slots collects date information. The user might ask `What's the weather meant to be tomorrow?`. You might have a root node that conditions on #forecast that might answer the user. However, because the user's input includes the word `tomorrow` and the reservation node with slots is being processed, your assistant assumes that the user is providing or updating the reservation date instead. *The current node always gets priority.* If you define a clear confirmation statement, such as, `Ok, setting the reservation date to tomorrow,` the user is more likely to realize there was a miscommunication and correct it.

If the user provides a value that is not expected by any of the slots, it might match an unrelated root node that the user never intended to digress to.

Be sure to do lots of testing as you configure the digression behavior.

When to use digressions instead of slot handlers : For general questions that users might ask, use a root node that allows digressions into it, processes the input, and then goes back to the flow that was in progress. For nodes with slots, try to anticipate the types of related questions users might want to ask when filling slots, and address them by adding handlers to the node.

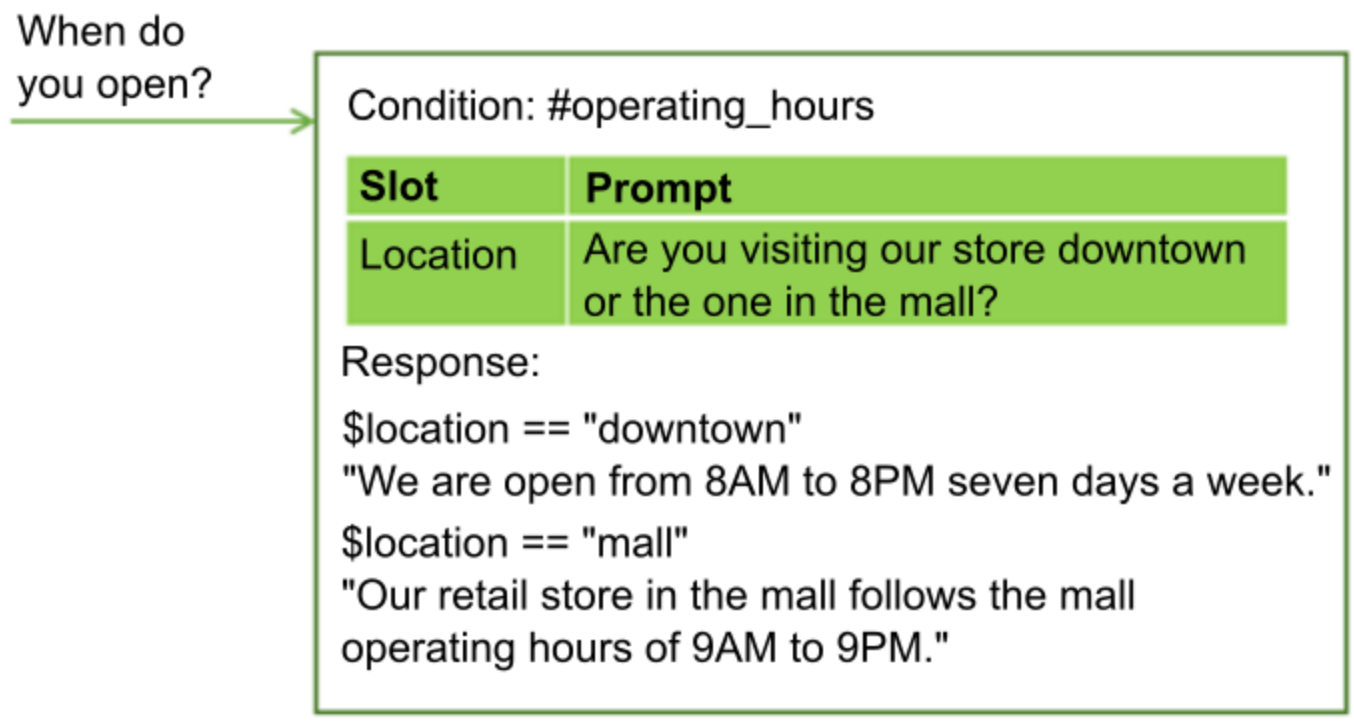
For example, if the node with slots collects the information that is required to complete an insurance claim, then you might want to add handlers that address common questions about insurance. However, for questions about how to get help, or your store locations, or the history of your company, use a root level node.

Gathering information with slots

Add slots to a dialog node to gather multiple pieces of information from a user within that node. Slots collect information at the user's pace. Details that a user provides up front are saved, and your assistant asks only for the missing details it needs to fulfill the request.

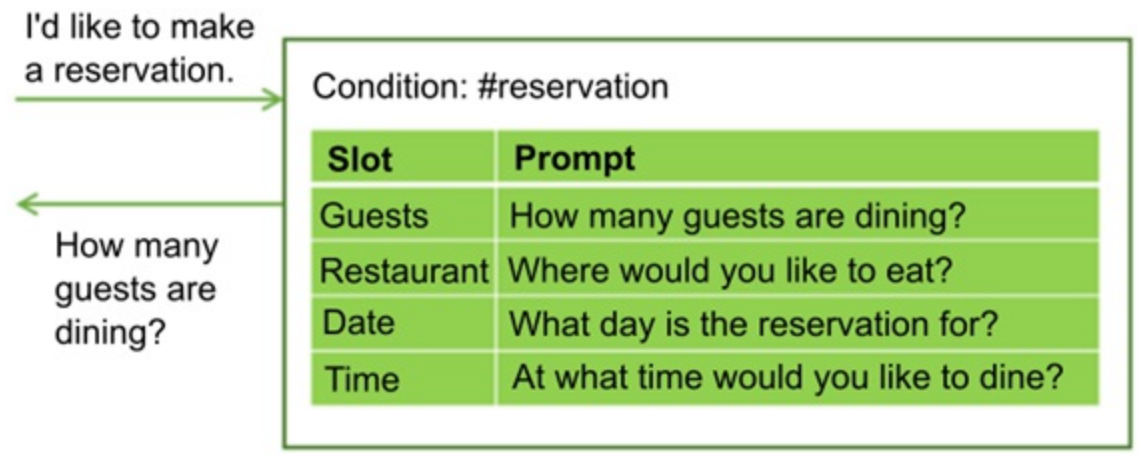
Why add slots?

Use slots to get the information you need before you can respond accurately to the user. For example, if users ask about operating hours, but the hours differ by store location, you might ask a follow-up question about which store location they plan to visit before you answer. You can then add response conditions that take the provided location information into account.



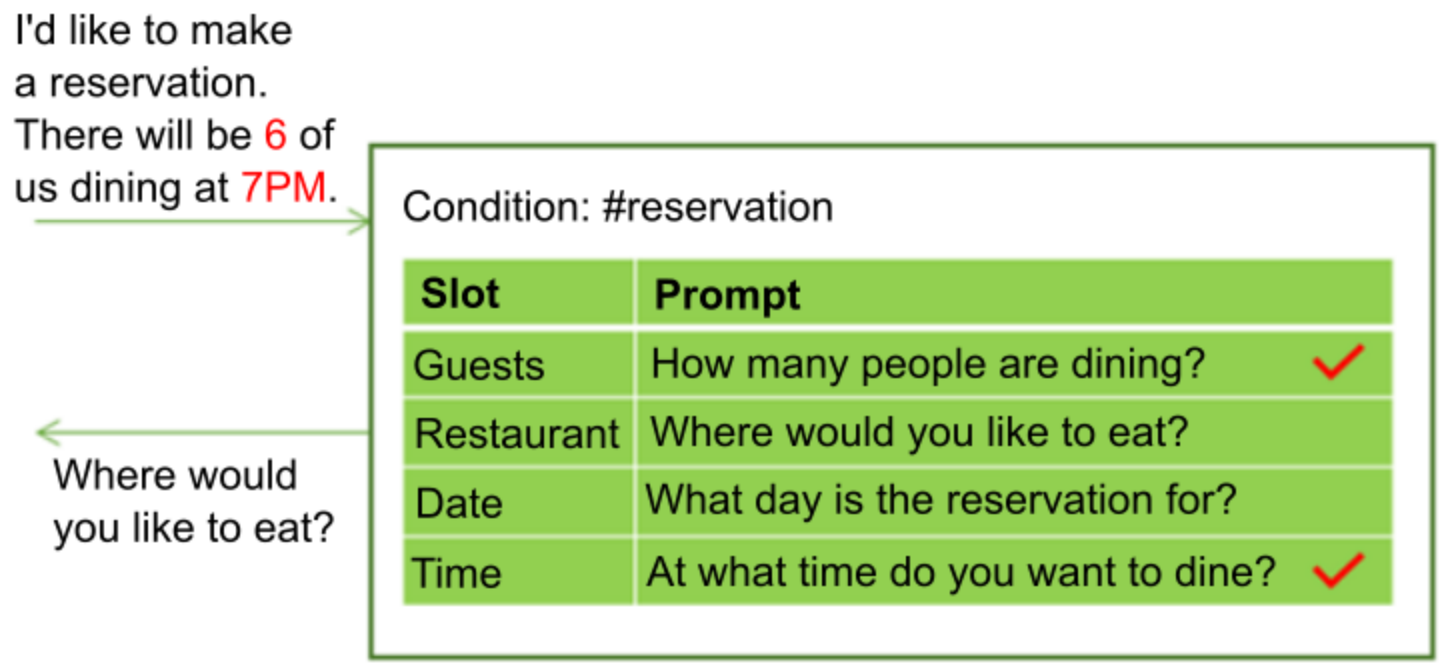
Ask for location

Slots can help you to collect multiple pieces of information that you need to complete a complex task for a user, such as making a dinner reservation.



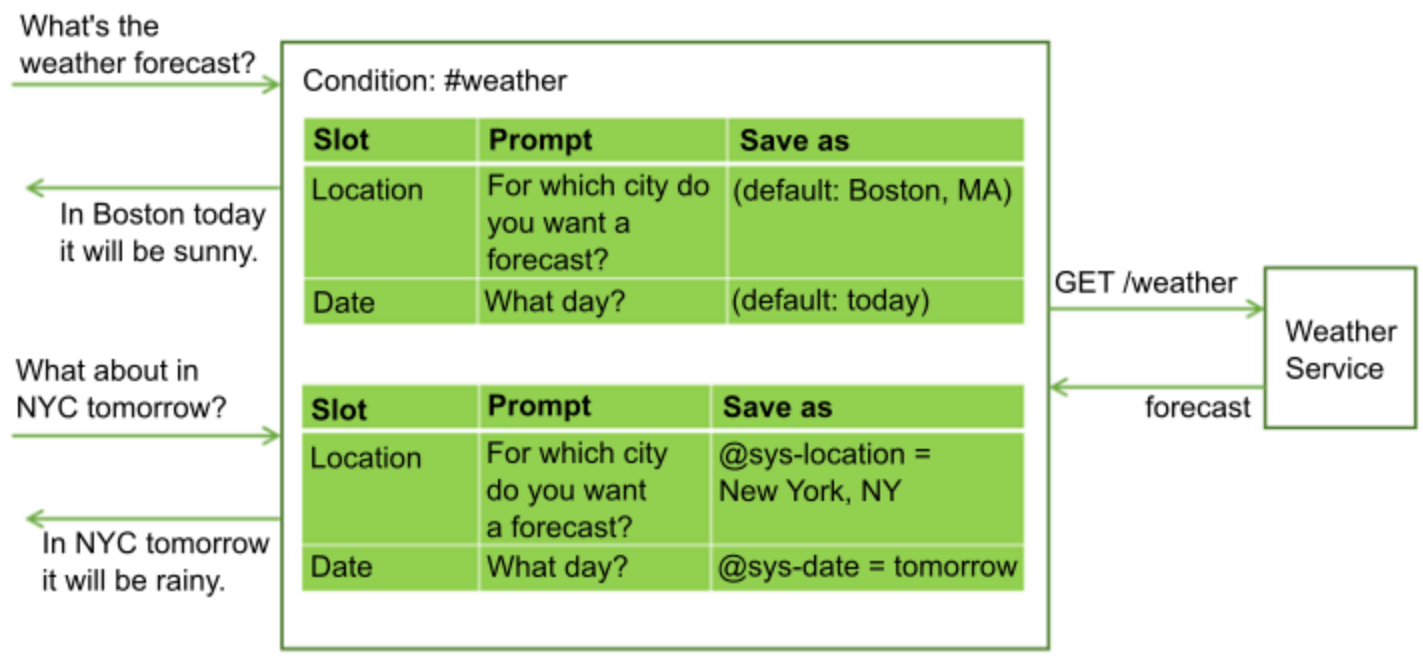
Four slots for reservation

The user might provide values for multiple slots at once. For example, the input might include the information `There will be 6 of us dining at 7 PM`. This one input contains two of the missing required values: the number of guests and time of the reservation. Your assistant recognizes and stores both of them, each one in its corresponding slot. It then displays the prompt that is associated with the next empty slot.



Two slots filled

Slots make it possible for your assistant to answer follow-up questions without having to reestablish the user's goal. For example, a user might ask for a weather forecast, then ask a follow-up question about weather in another location or on a different day. If you save the required forecast variables, such as location and day, in slots, then if a user asks a follow-up question with new variable values, you can overwrite the slot values with the new values provided, and give a response that reflects the new information. For more information about how to call an external service from a dialog, see Making programmatic calls from a dialog node.



Follow-up question

Using slots produces a more natural dialog flow between the user and your assistant, and is easier for you to manage than trying to collect the information by using many separate nodes.

Adding slots

1. Identify the units of information that you want to collect. For example, to order a pizza for someone, you might want to collect the following information:
   - Delivery time
   - Size
2. From the dialog node edit view, click **Customize**, and then set the **Slots** switch to **On**.

   For more information about the **Prompt for everything** checkbox, see Asking for everything at once.
3. Add a slot for each unit of required information.

   For each slot, specify these details:

   - **Check for**: Identify the type of information that you want to extract from the user's response to the slot prompt. In most cases, you check for entity values. In fact, the condition builder that is displayed suggests entities that you can check for. However, you can also check for an intent; type the intent name into the field. You can use AND and OR operators here to define more complex conditions.

     ⚠ **Important:** The *Check for* value is first used as a condition, but then becomes the value of the context variable that you name in the

> *Save it as* field. It specifies both **what to check for** and **what to save**. If you want to change how the value is saved, then add the expression that reformats the value to the *Save it as* field.

For example, if the entity is a pattern entity, such as `@email`, then append `.literal` to it. Adding `.literal` indicates that you want to capture the exact text that was entered by the user and was identified as an email address based on its pattern.

In some cases, you might want to use an expression to capture the value, but not apply the expression to what is saved. You can use one value in the *Check for* field to capture the value, and then open the JSON editor to change the value of the context variable, so it saves something else.

> **Note:** Any edit you make to a slot's context variable value in the JSON editor is not reflected in the **Check for** field after you exit the JSON editor. You must reopen the JSON editor to see what is saved as the context variable value.

Avoid checking for context variable values in the *Check for* field. Because the value you check for is also the value that is saved, a context variable in the condition can lead to unexpected behavior.

- **Save it as**: Provide a name for the context variable in which to store the value of interest from the user's response to the slot prompt.

  Do not reuse a context variable that is used elsewhere in the dialog. If the context variable has a value already, then the slot's prompt is not displayed. It is only when the context variable for the slot is null that the prompt for the slot is displayed.

- **Prompt**: Write a statement that elicits the piece of the information you need from the user. At this prompt, the conversation pauses and your assistant waits for the user to respond.

  If you want the prompt to be something other than a text response, you can change the response type by clicking the **Customize slot** ! [Customize slot] icon. Click **Text** to choose a different response type.

  Response type options:

  - Connect to human agent
  - Image
  - Option
  - Pause
  - Search skill Plus
  - Text

This table shows example slot values for a node that helps users place a pizza order by collecting two pieces of information, the pizza size and delivery time.

Check for	Save it as	Prompt	Follow-up if found	Follow-up if not found
@size	$size	What size pizza would you like?	$size it is.	What size did you want? We have small, medium, and large.
@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.

Example slots for pizza order

4. **Add slot value validation**: If you want different follow-up statements to be shown based on whether the user provides the information you need in response to the initial slot prompt, you can edit the slot and define the follow-up statements:

   Found: Displayed after the user provides the expected information.

   Not found: Displayed only if the information provided by the user is not understood, which means all of the following are true:

   - None of the active slots in the node are filled successfully
   - No slot handlers are understood
   - If digressions away are enabled for the node, no top-level dialog nodes are triggered as a digression from slot filling

   For information about how to define conditions and associated actions for *Found* and *Not found* responses, see Adding conditions to Found and Not found responses.

5. **Make a slot optional or disable it under certain conditions** . You can configure a slot in these ways:

- **Optional**: To make a slot optional, add a slot without a prompt. Your assistant does not ask the user for the information, but it does look for the information in the user input, and saves the value if the user provides it. For example, you might add a slot that captures dietary restriction information in case the user specifies any. However, you don't want to ask all users for dietary information since it is irrelevant in most cases.

Information	Check for	Save it as
Wheat restriction	@dietary	$dietary

Optional slot

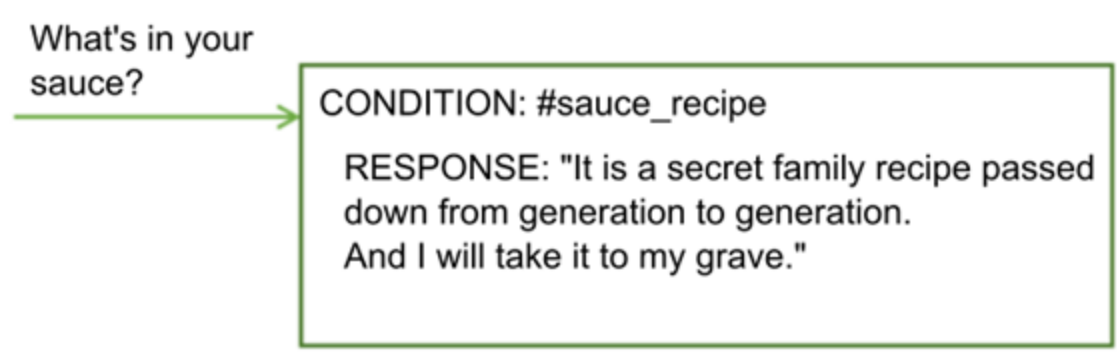
Tip: If you make a slot optional, reference its context variable in the node-level response text if you can word it such that it makes sense even if no value is provided for the slot. For example, you might word a summary statement like this, `I am ordering a $size $dietary pizza for delivery at $time.` The resulting text makes sense whether the dietary restriction information, such as `gluten-free` or `dairy-free`, is provided or not. The result is either, `I am ordering a large gluten-free pizza for delivery at 3:00PM.` or `I am ordering a large pizza for delivery at 3:00PM.`

- **Conditional**: If you want a slot to be enabled only under certain conditions, then you can add a condition to it. For example, if slot 1 asks for a meeting start time, slot 2 captures the meeting duration, and slot 3 captures the end time, then you might want to enable slot 3 only if a value for slot 2 is not provided. To make a slot conditional, edit the slot, and then from the **More** ⋮ menu, select **Enable condition**. Define the condition that must be met for the slot to be enabled.

  Tip: You can condition on the value of a context variable from an earlier slot because the order in which the slots are listed is the order in which they are evaluated. However, condition on a slot context variable that you can be confident contains a value when this slot is evaluated. The earlier slot must be a required slot, for example.

6. **Keep users on track**. You can define slot handlers that provide responses to questions users might ask during the interaction that are tangential to the purpose of the node.

   For example, the user might ask about the tomato sauce recipe or where you get your ingredients. To handle such off-topic questions, click the **Manage handlers** link and add a condition and response for each anticipated question.



Off-topic question

   After responding to the off-topic question, the prompt that is associated with the current empty slot is displayed.

   This condition is triggered if the user provides input that matches the slot handler conditions at any time during the dialog node flow up until the node-level response is displayed. See Handling requests to exit a process for more ways to use the slot handler.

7. **Add a node-level response**. The node-level response is not executed until after all of the required slots are filled. You can add a response that summarizes the information you collected. For example, `A $size pizza is scheduled for delivery at $time. Enjoy!`

   You can show an image or list of options as a response instead of a text response. See Response type options.

   If you want to define different responses based on certain conditions, click **Customize**, and then set the **Multiple responses** switch to **On**. For more information, see Conditional responses.

8. **Add logic that resets the slot context variables** . As you collect answers from the user per slot, they are saved in context variables. You can use the context variables to pass the information to another node or to an application or external service for use. However, after passing the information, you must set the context variables to null to reset the node so it can start collecting information again. You cannot null the context variables within the current node because your assistant doesn't exit the node until the required slots are filled. Instead, consider using one of the following methods:

   - Add processing to the external application that nulls the variables.
   - Add a child node that nulls the variables.
   - Insert a parent node that nulls the variables, and then jumps to the node with slots.

Slots usage tips

The following slot properties can help you check and set values in slot context variables.

Property name	Description
all_slots_filled	Evaluates to true only if all of the context variables for all of the slots in the node have been set. See Preventing a Found response from displaying when it is not needed for a usage example.
event.current_value	Current value of the context variable for this slot. See Replacing a slot context variable value for a usage example for this property and the event.previous_value property.
event.previous_value	Previous value of the context variable for this slot.
has_skipped_slots	True if any of the slots or slot handlers that are configured with a next step option that skips slots was processed. See Adding conditions to Found and Not found responses for more information about next step options for slots and Handling requests to exit a process for information about next step options for slot handlers.
slot_in_focus	Forces the slot condition to be applied to the current slot only. See Getting confirmation for more details. You can use this property to collect and store the exact words that are submitted by a customer. See Collecting summary information from the customer.

Slot properties

Consider using these approaches for handling common tasks.

- Asking for everything at once
- Capturing multiple values
- Reformatting values
- Dealing with zeros
- Getting confirmation
- Collecting summary information from the customer
- Replacing a slot context variable value
- Avoiding number confusion
- Adding conditions to Found and Not found responses
- Moving on after multiple failed attempts
- Preventing a Found response from displaying when it is not needed
- Handling requests to exit a process

Asking for everything at once

Include an initial prompt for the whole node that clearly tells users which units of information you want them to provide. Displaying this prompt first gives users the opportunity to provide all the details at once and not wait to be prompted for each piece of information one at a time.

For example, when the node is triggered because a customer wants to order a pizza, you can respond with the preliminary prompt, `I can take your pizza order. Tell me what size pizza you want and the time that you want it delivered.`

If the user provides even one piece of this information in their initial request, then the prompt is not displayed. For example, the initial input might be, `I want to order a large pizza.` When your assistant analyzes the input, it recognizes `large` as the pizza size and fills the **Size** slot with the value provided. Because one of the slots is filled, it skips displaying the initial prompt to avoid asking for the pizza size information again. Instead, it displays the prompts for any remaining slots with missing information.

From the Customize pane where you enabled the Slots feature, select the **Prompt for everything** checkbox to enable the initial prompt. This setting adds the **If no slots are pre-filled, ask this first** field to the node, where you can specify the text that prompts the user for everything.

Capturing multiple values

You can ask for a list of items and save them in one slot.

For example, you might want to ask users whether they want toppings on their pizza. To do so define an entity (@toppings), and the accepted values for it (pepperoni, cheese, mushroom, and so on). Add a slot that asks the user about toppings. Use the values property of the entity type to capture multiple values, if provided.

Check for	Save it as	Prompt	Follow-up if found	Follow-up if not found

@toppings.values	$toppings	Any toppings on that?	Great addition.	What toppings would you like? We offer ...

Multiple value slot

To reference the user-specified toppings later, use the `<? $entity-name.join(',') ?>` syntax to list each item in the toppings array and separate the values with a comma. For example, `I am ordering you a $size pizza with <? $toppings.join(',') ?> for delivery by $time.`

Reformatting values

Because you are asking for information from the user and need to reference their input in responses, consider reformatting the values so you can display them in a friendlier format.

For example, time values are saved in the `hh:mm:ss` format. You can use the JSON editor for the slot to reformat the time value as you save it so it uses the `hour:minutes AM/PM` format instead:

```
{
  "context":{
    "time": "<? @sys-time.reformatDateTime('h:mm a') ?>"
  }
}
```

Dealing with zeros

Using `@sys-number` in a slot condition is helpful for capturing any numbers that users specify in their input. However, it does not behave as expected when users specify the number (0) zero. Instead of treating zero as a valid number, the condition is evaluated to false, and your assistant prompts the user for a number again. To prevent this behavior, check for an `@sys-number` mention that is greater than or equal to zero in the slot condition.

To ensure that a slot condition that checks for number mentions deals with zeros properly, complete the following step:

1. Add `@sys-number >= 0` to the slot condition field, and then provide the context variable name and text prompt.

   What you check for in the input is also what is saved in the slot context variable. However, in this case, you want only the number (such as `5`) to be saved. You do not want to save `5 > = 0`. To change what is saved, you must edit the value of the context variable.

2. Open the slot to edit it by clicking the **Customize slot** icon. From the **Options** ⋮ menu, open the JSON editor.

3. Change the context variable value.

   The value looks like this:

   ```
   {
     "context": {
       "number": "@sys-number >= 0"
     }
   }
   ```

   Change it to look like this:

   ```
   {
     "context": {
       "number": "@sys-number"
     }
   }
   ```

4. Save your changes.

Note: The change that you made to the context variable value is not reflected in the **Check for** field. You must reopen the JSON editor to see what is saved.

If you do not want to accept a zero as the number value, then you can add a conditional response for the slot to check for zero, and tell the user that they must provide a number greater than zero. But, it is important for the slot condition to be able to recognize a zero when it is provided as input.

Getting confirmation

Add a slot after the others that asks the user to confirm that the information you collect is accurate and complete. The slot can look for responses that match the #yes or #no intent.

Check for	Save it as	Prompt	Follow-up if found	Follow-up if not found
#yes || #no	$confirmation	I'm going to order you a $size pizza for delivery at $time. Should I go ahead?	Your pizza is on its way!	see *Complex response*

Confirmation slot

Complex response Because users might include affirmative or negative statements at other times during the dialog (*Oh yes, we want the pizza delivered at 5pm*) or (*no guests tonight, let's make it a small*), use the `slot_in_focus` property to make it clear in the slot condition that you are looking for a Yes or No response to the prompt for this slot only.

```
(#yes || #no) && slot_in_focus
```

Tip: The `slot_in_focus` property always evaluates to a Boolean (true or false) value. Include it in a condition for which you want a Boolean result. Do not use it in slot conditions that check for an entity type and then save the entity value, for example.

In the **Not found** prompt, clarify that you are expecting the user to provide a Yes or No answer.

```
{
  "output":{
    "text": {
      "values": [
        "Respond with Yes to indicate that you want the order to
         be placed as-is, or No to indicate that you do not."
      ]
    }
  }
}
```

In the **Found** prompt, add a condition that checks for a No response (#no). When found, ask for the information all over again and reset the context variables that you saved earlier.

```
{
  "conditions": "#no",
  "output":{
    "text": {
      "values": [
        "Let's try this again. Tell me what size pizza
         you want and the time..."
      ]
    }
  },
  "context":{
    "size": null,
    "time": null,
    "confirmation": null
  }
}
```

Collecting summary information from the customer

You might want to prompt a user to supply free form text in a dialog node with slots that you can save and refer to later. To do so, follow these steps:

1. In the *Check for* field, add the following special property: `slot_in_focus`.
2. Optionally, change the context variable name for the slot in the *Save it as* field. For example, you might want to change it to something like `summary`.
3. In the *If not present, ask* field, ask the user to provide open-ended information. For example, `Can you summarize the problem?`
4. To store the input in the customer's exact words, edit what is saved by using the JSON editor.
5. Open the slot to edit it by clicking the **Customize slot** icon. From the **Options** ⋮ menu, open the JSON editor.
6. Change the context variable value.

   The value looks like this:

```
{
  "context": {
    "summary": "slot_in_focus"
  }
}
```

Change the value that is saved in the `$summary` context variable. You want to capture and store whatever text the user submits. To do so, use a SpEL expression that captures the input text.

```
{
  "context": {
    "summary": "<? input.text ?>"
  }
}
```

7. Save your changes.

Note: The change that you made to the context variable value is not reflected in the *Check for* field. You must reopen the JSON editor to see what is saved.

Replacing a slot context variable value

If at any time before the user exits a node with slots, the user provides a new value for a slot, then the new value is saved in the slot context variable, replacing the previous value. Your dialog can acknowledge explicitly that this replacement occurred by using special properties that are defined for the Found condition:

- `event.previous_value`: Previous value of the context variable for this slot.
- `event.current_value`: Current value of the context variable for this slot.

For example, your dialog asks for a destination city for a flight reservation. The user provides `Paris`. You set the $destination slot context variable to *Paris*. Then, the user says `Oh wait. I want to fly to Madrid instead`. If you set up the Found condition as follows, then your dialog can handle this type of change gracefully.

When the user responds, if @destination is found:

```
Condition: (event.previous_value != null) &&
        (event.previous_value != event.current_value)
   Response: Ok, updating destination from
   <? event.previous_value ?> to <? event.current_value ?>.
Response: Ok, destination is $destination.
```

This slot configuration enables your dialog to react to the user's change in destination by saying, `Ok, updating the destination from Paris to Madrid.`

Avoiding slot filling confusion

When a user input is evaluated, the slot with the first slot condition to match it is filled only. Test for the following possible causes of misinterpretation, and address them:

- **Problem**: The same entity is used in more than one slot.

  For example, `@sys-date` is used to capture the departure date in one slot and arrival date in another.

  Solution: Use slot found conditions that get clarification from the user about which date you are saving in a slot before you save it.

- **Problem**: A term fully or partially matches the entities in more than one slot condition.

  For example, if one slot captures a product ID (`@id`) with a syntax like `GR1234` and another slot captures a number (`@number`), such as `1234`, then user input that contains an ID, such as `BR3344` might get claimed by the `@number` slot as a number reference and fill the `$number` context variable with `3344`.

  Solution: Place the slot with the entity condition that captures the longer pattern (@id) higher in the list of slots than the condition that captures the shorter pattern (@number).

- **Problem**: A term is recognized as more than one system entity type.

  For example, if the user enters *May 2*, then your assistant recognizes both the `@sys-date` (2017-05-02) and `@sys-number` (2) entities.

  Solution: In logic that is unique to the slots feature, when two system entities are recognized in a single user input, the one with the larger span is

used. Therefore, even though your assistant recognizes both system entities in the text, only the system entity with the longer span (`@sys-date` with `2017-05-02`) is registered and applied to the slot.

> **Note:** This workaround is not necessary if you are using the revised system entities. With the updated entities, a date reference is considered to be a `@sys-date` mention only, and is not also treated as a `@sys-number` mention. For more information, see System entities.

Adding conditions to Found and Not found responses

For each slot, you can use conditional responses with associated actions to help you extract the information that you need from the user. To do so, follow these steps:

1. Click the **Customize slot** icon for the slot to which you want to add conditional Found and Not found responses.
2. From the **More** ⋮ menu, select **Enable conditional responses**.
3. Enter the condition and the response to display if the condition is met.

   Found example: A slot is expecting the time for a dinner reservation. You might use @sys-time in the *Check for* field to capture it. To prevent an invalid time from being saved, you can add a conditional response that checks whether the time provided is before the restaurant's last seating time, for example `@sys-time.after('21:00:00')`. The corresponding response might be something like *Our last seating is at 9PM.*

   Not found example: The slot is expecting a @sys-number entity for the number of pizzas in a takeout order. The Not found condition of `true` might be used to display a message that prompts the user, just in case a number isn't provided in the conversation (for example, `We need to know how many pizzas you want.`).
4. If you want to customize what happens next if the condition is met, then click the **Customize handler** icon.

   For Found responses that are displayed when the user provides a value that matches the value type that is specified in the Check for field, you can choose one of these actions:

   - **Move on (Default)**: Instructs your assistant to move on to the next empty slot after displaying the response. In the associated response, assure the user that their input was understood. For example, *Ok. You want to schedule it for $date.*
   - **Clear slot and prompt again** : If you are using an entity in the *Check for* field that might pick up the wrong value, add conditions that detect any likely misinterpretations, and use this action to clear the current slot value and prompt for the correct value.
   - **Skip to response**: If the condition you define is met, you no longer need to fill any of the remaining slots in this node, choose this action to skip the remaining slots and go directly to the node-level response next. For example, you might add a condition that checks whether the user's age is under 16. If so, you might skip the remaining slots that ask questions about the user's driving record.

   For Not found responses (that are displayed when the user does not provide a valid value), you can choose one of these actions:

   - **Wait for user input (Default)** : Pauses the conversation and your assistant waits for the user to respond. In the simplest case, the text you specify here can more explicitly state the type of information you need the user to provide. If you use this action with a conditional response, be sure to word the conditional response such that you clearly state what was wrong with the user's answer and what you expect them to provide instead.

   - **Prompt again**: After the Not found response, your assistant repeats the slot prompt again and waits for the user to respond. If you use this action with a conditional response, the response can merely explain what was wrong about the answer the user provided. It does not need to reiterate the type of information you want the user to provide because the slot prompt typically explains that.

     > **Tip:** If you choose this option, consider adding at least one variation of the Not found response so that the user does not see the exact same text more than once. Take the opportunity to use different wording to explain to the user what information you need them to provide and in what format.

   - **Skip this slot**: Instructs your assistant to stop trying to fill the current slot, and instead, move on to the prompt for the next empty slot. This option is useful in a slot where you want to both make the slot optional and to display a prompt that asks the user for information. For example, you might have a @seating entity that captures restaurant seating preferences, such as *outside*, *near the fireplace*, or *private*. You can add a slot that prompts the user with, *Do you have any seating preferences?* and checks for `@seating.values`. If a valid response is provided, it saves the preference information to `$seating_preferences`. However, by choosing this action as the Not found response next step, you instruct your assistant to stop trying to fill this slot if the user does not provide a valid value for it.

   - **Skip to response**: If the condition you define is met, you no longer need to fill any of the remaining slots in this node, choose this action to skip the remaining slots and go directly to the node-level response next. For example, if when capturing the one-way flight information, the slot prompt is, *Are you buying round-trip tickets?* the Not found condition can check for #No. If #No is found, use this option to skip the remaining slots that capture information about the return flight, and go straight to the node-level response instead.

Click **Back** to return to the edit view of the slot.

5. To add another conditional response, click **Add a response**, and then enter the condition and the response to display if the condition is met.

   Be sure to add at least one response that is used no matter what. You can leave the condition field blank for this catchall response. Your assistant automatically populates the empty condition field with the `true` special condition.

6. Click **Save** to save your changes, close the edit view of the slot, and return to the edit view of the node.

Moving on after multiple failed attempts

You can provide users with a way to exit a slot if they cannot answer it correctly after several attempts by using Not found conditional responses. In the catchall response, open the JSON editor to add a counter context variable that keeps track of the number of times the Not found response is returned. In an earlier node, be sure to set the initial counter context variable value to 0.

In this example, your assistant asks for the pizza size. It lets the user answer the question incorrectly 3 times before applying a size (medium) to the variable for the user. (You can include a confirmation slot where users can always correct the size when they are asked to confirm the order information.)

Check for: @size Save it as: $size Not found catchall condition:

```
{
  "output": {
    "text": {
      "values": [
        "What size did you want? We have small, medium, and large."
      ],
      "selection_policy": "sequential"
    }
  },
  "context": {
    "counter": "<? context['counter'] + 1 ?>"
  }
}
```

To respond differently after 3 attempts, add another Not found condition like this:

```
{
  "conditions": "$counter > 1",
  "output": {
    "text": {
      "values": [
        "We will bring you a size medium pizza."
      ]
    }
  },
  "context": {
    "size": "medium"
  }
}
```

This Not found condition is more precise than the Not found catchall condition, which defaults to `true`. Therefore, you must move this response so it comes before the original conditional response or it will never be triggered. Select the conditional response and use the up arrow to move it up.

Preventing a Found response from displaying when it is not needed

If you specify Found responses for multiple slots, then if a user provides values for multiple slots at once, the Found response for at least one of the slots will be displayed. You probably want either the Found response for all of them or none of them to be returned.

To prevent Found responses from being displayed, you can do one of the following things to each Found response:

- Add a condition to the response that prevents it from being displayed if particular slots are filled. For example, you can add a condition, like `!($size && $time)`, that prevents the response from being displayed if the $size and $time context variables are both provided.
- Add the `!all_slots_filled` condition to the response. This setting prevents the response from being displayed if all of the slots are filled. Do not use this approach if you are including a confirmation slot. The confirmation slot is also a slot, and you typically want to prevent the Found responses from being displayed before the confirmation slot itself is filled.

Handling requests to exit a process

Add at least one slot handler that can recognize it when a user wants to exit the node.

For example, in a node that collects information to schedule a pet grooming appointment, you can add a handler that conditions on the #cancel intent,

which recognizes utterances such as, `Forget it. I changed my mind`.

1. In the JSON editor for the handler, fill all of the slot context variables with dummy values to prevent the node from continuing to ask for any that are missing. And in the handler response, add a message such as, `Ok, we'll stop there. No appointment will be scheduled.`
2. Choose what action you want your assistant to take next from the following options:
   - **Prompt again (Default)**: Displays the prompt for the slot that the user was working with just before asking the off-topic question.
   - **Skip current slot**: Displays the prompt associated with the slot that comes after the slot that the user was working with just before asking the off-topic question. And your assistant makes no further attempts to fill the skipped slot.
   - **Skip to response**: Skips the prompts for all remaining empty slots including the slot the user was working with just before asking the off-topic question.
3. In the node-level response, add a condition that checks for a dummy value in one of the slot context variables. If found, show a final message such as, `If you decide to make an appointment later, I'm here to help.` If not found, it displays the standard summary message for the node, such as `I am making a grooming appointment for your $animal at $time on $date.`

Here's a sample of JSON that defines a handler for the pizza example. Note that, as described earlier, the context variables are all being set to dummy values. In fact, the `$size` context variable is being set to `dummy`. This $size value triggers the node-level response to show the appropriate message and exit the slots node.

```
{
"conditions": "#cancel",
 "output": {
   "text": {
     "values": [
       "Ok, we'll stop there. No pizza delivery will be scheduled."
     ],
    "selection_policy": "sequential"
    }
  },
"context": {
   "time": "12:00:00",
   "size": "dummy",
   "confirmation":"true"
}
}
```

Important: Take into account the logic used in conditions that are evaluated before this condition so you can build distinct conditions. When a user input is received, the conditions are evaluated in the following order:

- Slot handlers in the order they are listed.
- First slot condition.
- Current slot level Found conditions.
- If digressions away are allowed, root level node conditions are checked for a match (except the final `anything else` node in the dialog tree root or a root folder).
- Current slot level Not found conditions.
- Final `anything else` node condition.

Tip: Be careful about adding conditions that always evaluate to true (such as the special conditions, `true` and `anything_else`) as slot handlers. Per slot, if the slot handler evaluates to true, then the Not found condition is skipped entirely. So, using a slot handler that always evaluates to true effectively prevents the Not found condition for every slot from being evaluated.

For example, you groom all animals except cats. For the Animal slot, you might be tempted to use the following slot condition to prevent `cat` from being saved in the Animal slot:

```
Check for @animal && !@animal:cat, then save it as $animal.
```

And to tell users that you do not accept cats, you might specify the following value in the Not found condition of the Animal slot:

```
If @animal:cat then, "I'm sorry. We do not groom cats."
```

While logical, if you also define an #exit slot handler, then - given the order of condition evaluation - this Not found condition will likely never get triggered. Instead, you can use this slot condition:

```
Check for @animal, then save it as $animal.
```

And to deal with a possible `cat` response, add this value to the Found condition:

```
If @animal:cat then, "I'm sorry. We do not groom cats."
```

In the JSON editor for the Found condition, reset the value of the $animal context variable because it is currently set to cat and should not be.

```
{
  "output":{
    "text": {
      "values": [
        "I'm sorry. We do not groom cats."
      ]
    }
  },
  "context":{
    "animal": null
  }
}
```

Slots examples

To access JSON files that implement different common slot usage scenarios, go to the community repo in GitHub.

To explore an example, download one of the example JSON files, and then import it as a new dialog skill. From the Dialog tab, you can review the dialog nodes to see how slots were implemented to address different use cases.

Defining information to look for in customer input

Entities represent information in the user input that is relevant to the user's purpose.

If intents represent verbs (the task a user wants to do), entities represent nouns (the object of, or the context for, that task). For example, when the *intent* is to get a weather forecast, the relevant location and date *entities* are required before the application can return an accurate forecast.

Recognizing entities in the user's input helps you to craft more useful, targeted responses. For example, you might have a `#buy_something` intent. When a user makes a request that triggers the `#buy_something` intent, the assistant's response should reflect an understanding of what the *something* is that the customer wants to buy. You can add a `@product` entity, and then use it to extract information from the user input about the product that the customer is interested in. (The `@` prefix helps to clearly identify it as an entity.)

You can add multiple responses to your dialog tree with wording that differs based on the `@product` value that is detected in the user's request.

Entity evaluation overview

Your assistant detects entities in the user input by using one of the following evaluation methods:

- Dictionary-based method
- Annotation-based method

Dictionary-based method

Your assistant looks for terms in the user input that match the values, synonyms, or patterns you define for the entity.

- **Synonym entity**: You define a category of terms as an entity (`color`), and then one or more values in that category (`blue`). For each value you specify a bunch of synonyms (`aqua` , `navy`).

  At run time, your assistant recognizes terms in the user input that exactly match the values or synonyms that you defined for the entity as mentions of that entity.

- **Pattern entity**: You define a category of terms as an entity (`contact_info`), and then one or more values in that category (`email`). For each value, you specify a regular expression that defines the textual pattern of mentions of that value type. For an `email` entity value, you might want to specify a regular expression that defines a `text@text.com` pattern.

  At run time, your assistant looks for patterns that match your regular expression in the user input, and identifies any matches as mentions of that entity.

- **System entity**: Synonym entities that are prebuilt for you by IBM. They cover commonly used categories, such as numbers, dates, and times. You

simply enable a system entity to start using it.

Annotation-based method

When you define an annotation-based entity, which is also referred to as a contextual entity, a model is trained on both the *annotated term* and the *context* in which the term is used in the sentence you annotate. This contextual entity model enables your assistant to calculate a confidence score that identifies how likely a word or phrase is to be an instance of an entity, based on how it is used in the user input.

- **Contextual entity**: First, you define a category of terms as an entity (`product`). Next, you go to the *Intents* page and mine your existing intent user examples to find any mentions of the entity, and label them as such. For example, you might go to the `#buy_something` intent, and find a user example that says, `I want to buy a Coach bag` . You can label `Coach bag` as a mention of the `@product` entity.

  For training purposes, the term `Coach bag` is added as a value of the `@product` entity.

  At run time, your assistant evaluates terms based on the context in which they are used in the sentence only. If the structure of a user request that mentions the term matches the structure of an intent user example in which a mention is labeled, then your assistant interprets the term to be a mention of that entity type. For example, the user input might include the utterance `I want to buy a Gucci bag` . Due to the similarity of the structure of this sentence to the user example that you annotated (`I want to buy a Coach bag`), your assistant recognizes `Gucci bag` as a `@product` entity mention.

  When a contextual entity model is used for an entity, your assistant does *not* look for exact text or pattern matches for the entity in the user input, but focuses instead on the context of the sentence in which the entity is mentioned.

  If you choose to define entity values by using annotations, add at least 10 annotations per entity to give the contextual entity model enough data to be reliable.

Creating entities

1. Click **Entities**.
2. Click **Create entity**.

   You can also click **System entities** to select from a list of common entities, provided by IBM, that can be applied to any use case. See Enabling system entities for more detail.
3. In the **Entity name** field, type a descriptive name for the entity.

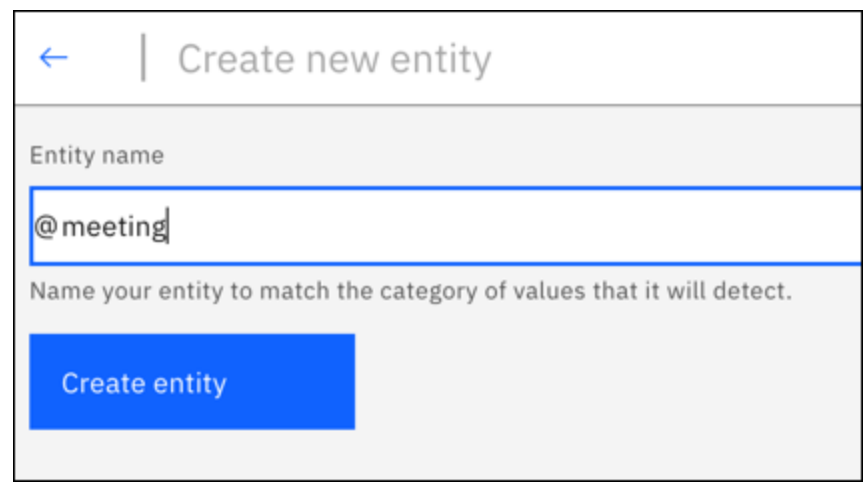
   The entity name can contain letters (in Unicode), numbers, underscores, and hyphens. For example:

   - `@location`
   - `@menu_item`
   - `@product`

   Do not include spaces in the name. The name cannot be longer than 64 characters. Do not begin the name with the string `sys-` because it is reserved for system entities.

   > **Tip:** The at sign prefix `@` is added to the entity name automatically to identify the term as an entity. You do not need to add it.

4. Click **Create entity**.



Create entity

5. For this entity, choose whether you want your assistant to use a dictionary-based or annotation-based approach to find mentions of it, and then follow the appropriate procedure.

   For each entity that you create, choose one entity type to use only. As soon as you add an annotation for an entity, the contextual model is initialized and becomes the primary approach for analyzing user input to find mentions of that entity. The context in which the mention is used in the user input takes precedence over any exact matches that might be present. For more information, see Entity evaluation overview.

Adding dictionary-based entities

Dictionary-based entities are used to define specific terms, synonyms, or patterns. At run time, your assistant finds entity mentions only when a term in the user input exactly matches (or closely matches if fuzzy matching is enabled) the value or one of its synonyms.

1. In the **Value name** field, type a value. For example, for the `@city` entity, you might type `New York City`.

   An entity value can be any string up to 64 characters in length.

   Important: Don't include sensitive or personal information in entity names or values. The names and values can be included in URLs in an app.

2. Add synonyms for the value. For example, you might add `NYC` and `The Big Apple` as synonyms for `New York City`.

   A synonym can be any string up to 64 characters in length.

   If you want to define a pattern for your assistant to look for in user input, such as a product order number or email address, define a pattern value instead. See Adding entities that recognize patterns for more details.

   Note: You can add *either* synonyms or patterns for a single entity value, not both.

3. If you want your assistant to recognize terms with syntax that is similar to the entity value and synonyms you specify, but without requiring an exact match, set the **Fuzzy Matching** switch to **On**.

   For example, if you add `apple` as a value for a `@fruit` entity, and a user enters `apples` or `appel`, if fuzzy matching is enabled, your assistant recognizes the word as a `@fruit` mention. For more information, see How fuzzy matching works.

4. Click **Add value** and repeat the process to add more entity values.

   > **Tip:** If you are adding many values, one after another, press **Shift+Enter** to finish adding the current value, and keep focus in the value field so you can add the next value.

5. After you add the entity values, click ← to finish creating the entity.

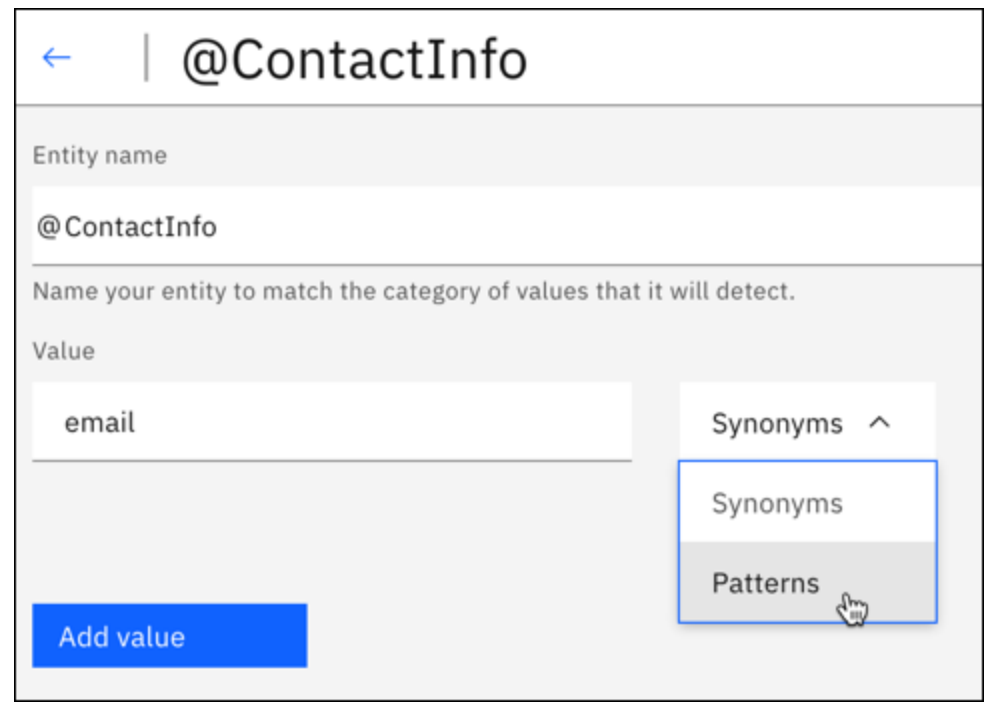
The entity that you created is added, and the system trains itself on the new data.

Adding entities that recognize patterns

You can create an entity that looks for patterns in user input. For example, you can look for mentions of an email address by looking for occurrences of the pattern `{word}+@+{word}+.com`. Or, you might have product order numbers that follow a specific format, such as `TWEX3433JKL`. You can create a pattern to look for strings with that syntax in the user utterance.

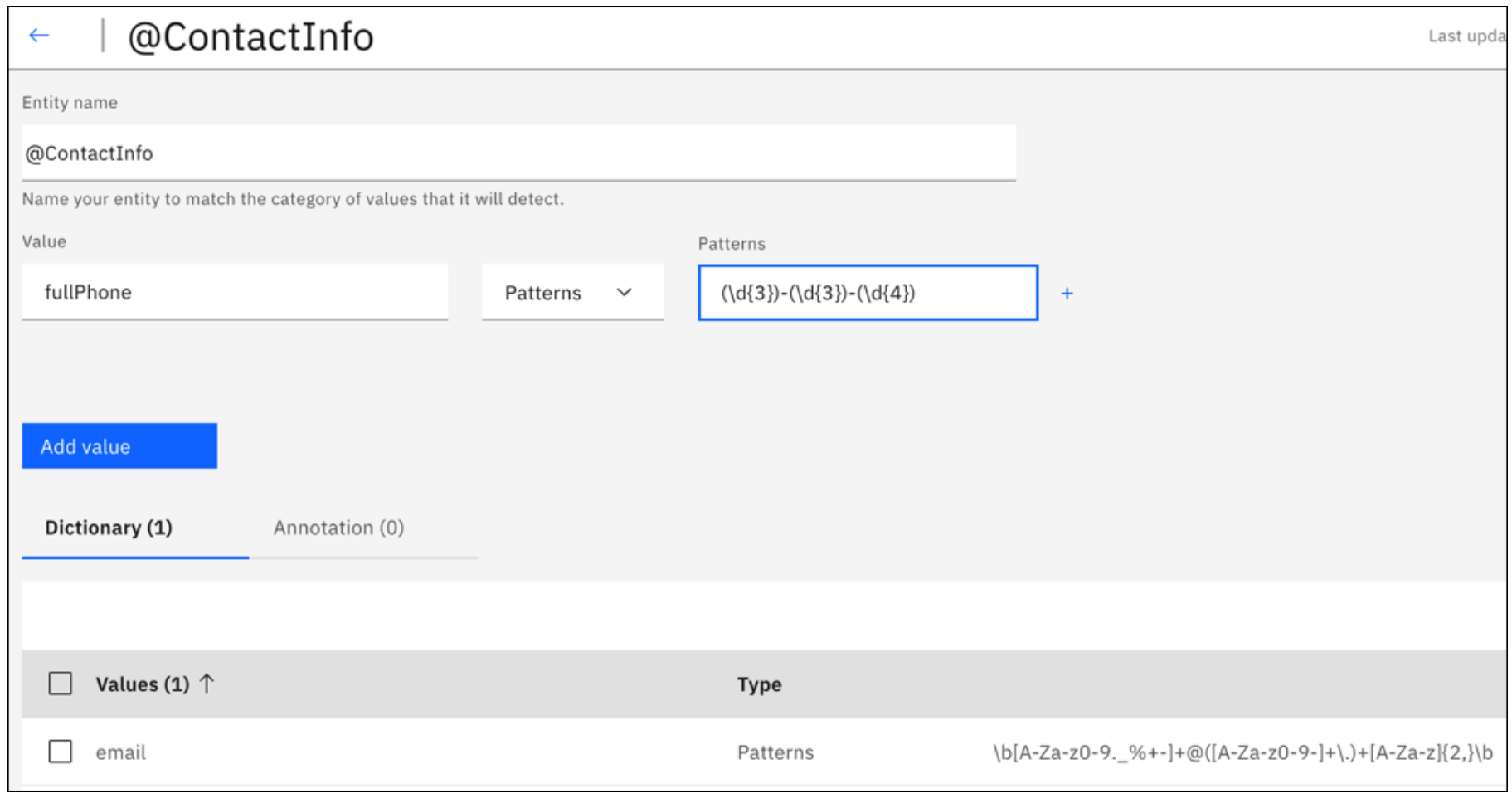
To add an entity that recognizes a pattern:

1. Follow the standard procedure to create a dictionary-based entity, but select **Patterns** from the *Type* menu instead of *Synonyms*.



Choose Patterns type

2. Add a regular expression that defines the pattern you want to look for.
   - For each entity value, there is a maximum of up to 5 patterns.
   - Each pattern (regular expression) is limited to 512 characters.



Patterns entity

Follow these syntax rules:

- Entity patterns can't contain:
  - Positive repetitions (for example `x*+`)
  - Backreferences (for example `\g1`)
  - Conditional branches (for example `(?(cond)true)`)
- When a pattern entity starts or ends with a Unicode character, and includes word boundaries, for example `\bš\b`, the pattern match does not match the word boundary correctly. In this example, for input `š zkouška`, the match returns `Group 0: 6-7 š` (`š zkou` `š` `ka`), instead of the correct `Group 0: 0-1 š` (`š` `zkouška`).

  The regular expression engine is loosely based on the Java regular expression engine. You see an error if you try to upload an unsupported pattern, either by using the API or from within the Watson Assistant user interface.

For example, for entity *ContactInfo*, the patterns for phone, email, and website values can be defined as follows:

- Phone
  - `localPhone`: `(\d{3})-(\d{4})`, for example, 426-4968
  - `fullUSphone`: `(\d{3})-(\d{3})-(\d{4})`, for example, 800-426-4968
  - `internationalPhone`: `^(\(?\+?[0-9]*\)?)?[0-9_\- \(\)]*$`, for example, +44 1962 815000
- Email
  - `email`: `\b[A-Za-z0-9._%+-]+@([A-Za-z0-9-]+\.)+[A-Za-z]{2,}\b`, for example, name@ibm.com
- Website
  - `website`: `(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$`, for example, https://www.ibm.com

3. Click **Add value** and repeat the process to add more entity values.

When you use pattern entities to find patterns in user input, you often need a way to store the part of the user input text that matches the pattern. To do so, you can use a context variable. For more information, see Defining a context variable.

For example, your dialog might ask users for their email address. The dialog node condition contains a condition similar to `@contactInfo:email`. You can use the following syntax in the dialog node's response section to define a context variable that captures and stores the user's email address text:

Variable	Value
email	`<? @contactInfo.literal ?>`

Saving a pattern

This syntax indicates that you want to find the part of the user input that matches the email pattern and save that subset of text into a context variable named `email`.

Capture groups

For regular expressions, any part of a pattern inside a pair of normal parentheses is captured as a group. For example, the entity `@ContactInfo` has the pattern `fullUSphone` that contains three captured groups:

- `(\d{3})` - US area code
- `(\d{3})` - Prefix
- `(\d{4})` - Line number

Grouping can be helpful if, for example, you want your assistant to ask for a phone number, and then use only the area code in a response.

To assign the user-entered area code as a context variable, use the following syntax in the dialog node's response section to capture the group match:

Variable	Value
area_code	`<? @ContactInfo.groups[1] ?>`

Saving a capture group

For more information about using capture groups in your dialog, see Storing and recognizing entity pattern groups in input.

How fuzzy matching works

Fuzzy matching is available for specific languages. For more information, see Supported languages.

Fuzzy matching has these components:

- *Stemming* - The feature recognizes the stem form of entity values that have several grammatical forms. For example, the stem of `bananas` would be `banana`, while the stem of `running` would be `run`.
- *Misspelling* - The feature is able to map user input to the appropriate corresponding entity despite the presence of misspellings or slight syntactical differences. For example, if you define `giraffe` as a synonym for an animal entity, and the user input contains the terms `giraffes` or `girafe`, the fuzzy match can map the term.
- *Partial match* - With partial matching, the feature automatically suggests substring-based synonyms present in the user-defined entities, and assigns a lower confidence score as compared to the exact entity match.

  Note: The partial match component is supported only in English-language dialog skills.

For English, fuzzy matching prevents the capturing of some common, valid English words as fuzzy matches for an entity. This feature uses standard English dictionary words. You can also define an English entity value and synonym, and fuzzy matching matches only your defined entity value or synonym. For example, fuzzy matching might match the term `unsure` with `insurance`; but if you define `unsure` as a value or synonym for an entity like `@option`, then `unsure` always matches to `@option`, and not to `insurance`.

Important: Interactions between the stemming and misspelling fuzzy matching features are not allowed. Specifically, if either an entity or the input is stemmed, misspelling fuzzy matching does not work. For example, assume that the entity is `@lending` and the input word is `pending`. During entity stemming, `@lending` produces `lend`. During input stemming, `pending` produces `pend`. In this case, `lend` does not match to `pend` because the entity and input were stemmed. This change applies to only the English language.

Adding contextual entities

Use annotation-based entities to annotate occurrences of the entity in sample sentences to teach your assistant about the context in which the entity is typically used.

To train a contextual entity model, you can take advantage of your intent examples, which provide sentences to annotate.

Note: This feature is generally available in English-language dialog skills and is available as a beta feature in French-langage dialog skills. For more information, see Supported languages.

Note: Using an intent's user examples to define contextual entities does not affect the classification of that intent. However, entity mentions that you label are also added to that entity as synonyms. And intent classification does use synonym mentions in intent user examples to establish a weak reference between an intent and an entity.

1. Click **Intents**.

2. Click an intent to open it.
3. Click **Annotate entities**, and then review the intent examples for potential entity mentions.
4. Click any word, words, or punctuation that is part of a single entity mention from the intent examples.

   In this example, `mobile phones` is the entity mention.

Review intent examples

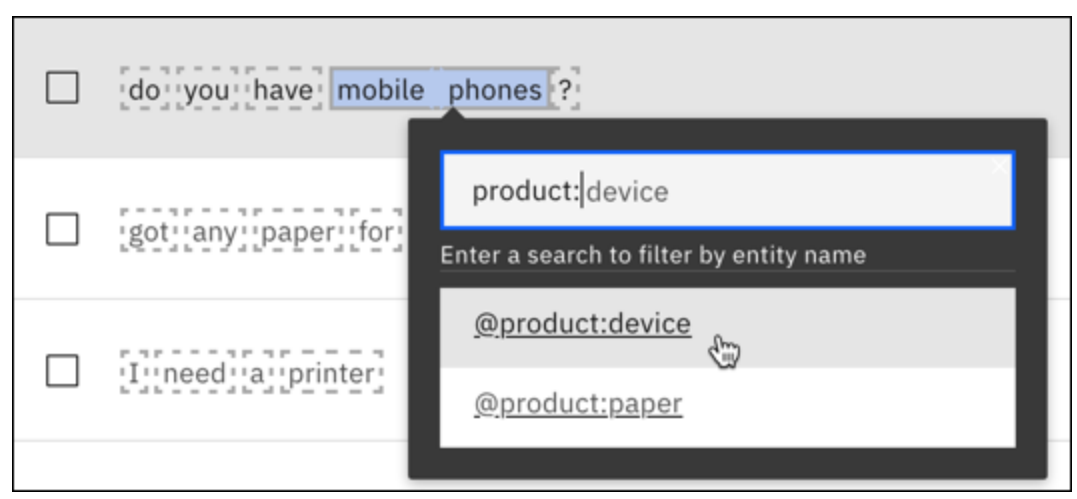
   A Search box opens that you can use to search for the entity that the highlighted word or phrase is a mention of.

Search box

5. Enter the entity name to search for. You do not need to include the starting `@` symbol.

   Do one of the following things:

   - If the entity has any existing entity values, they are displayed for informational purposes only. You are adding the annotation to the entity, not to any specific entity value.
   - If you want to teach the model that the mention is synonymous with an existing entity value, add a colon (`:`) after the entity name to show a list of entity values. Choose an entity value from the list that is displayed. For example, `@product:device`.



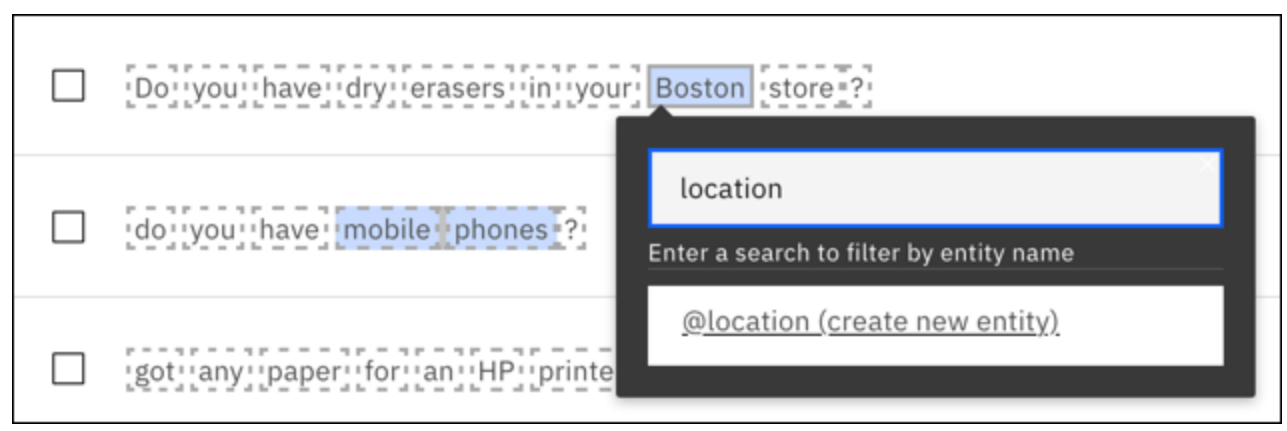
Show list of entity values

6. Select the entity or entity and value to which you want to add the annotation.

   In this example, `mobile phones` is being added as an annotation for the `@product` entity value and as a synonym for the `@product:device` entity value.

> ⚠ **Important:** Create *at least* 10 annotations for each contextual entity. More annotations are recommended for production use.

7. If none of the entities are appropriate, you can create a new entity by adding its name. Then, choose the **{entity_name}(create new entity)** option from the list.

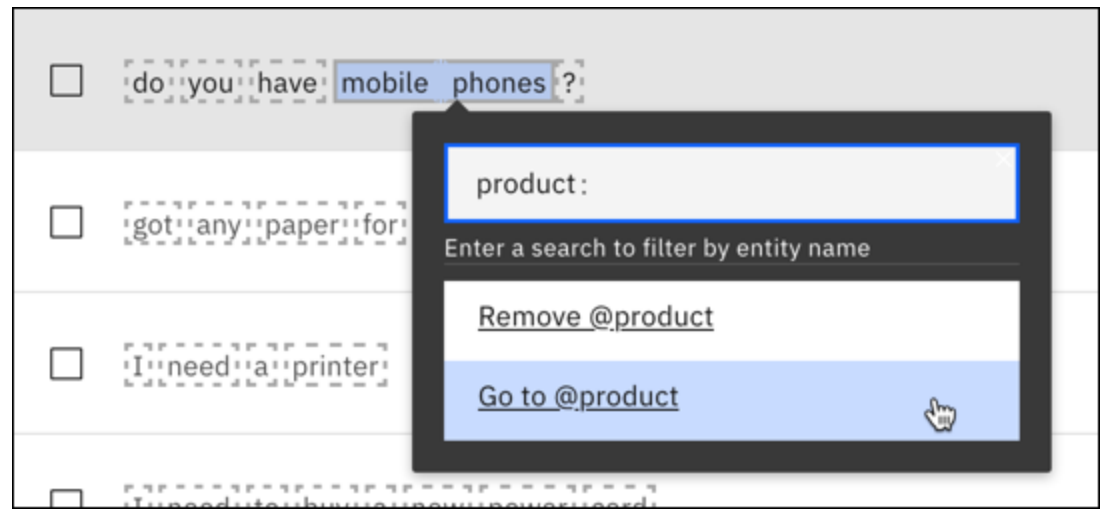


Create new entity

8. Repeat this process for each entity mention that you want to annotate.

⚠ **Important:** Be sure to annotate every mention of an entity type that occurs in any user examples that you edit. See What you don't annotate matters for more details.

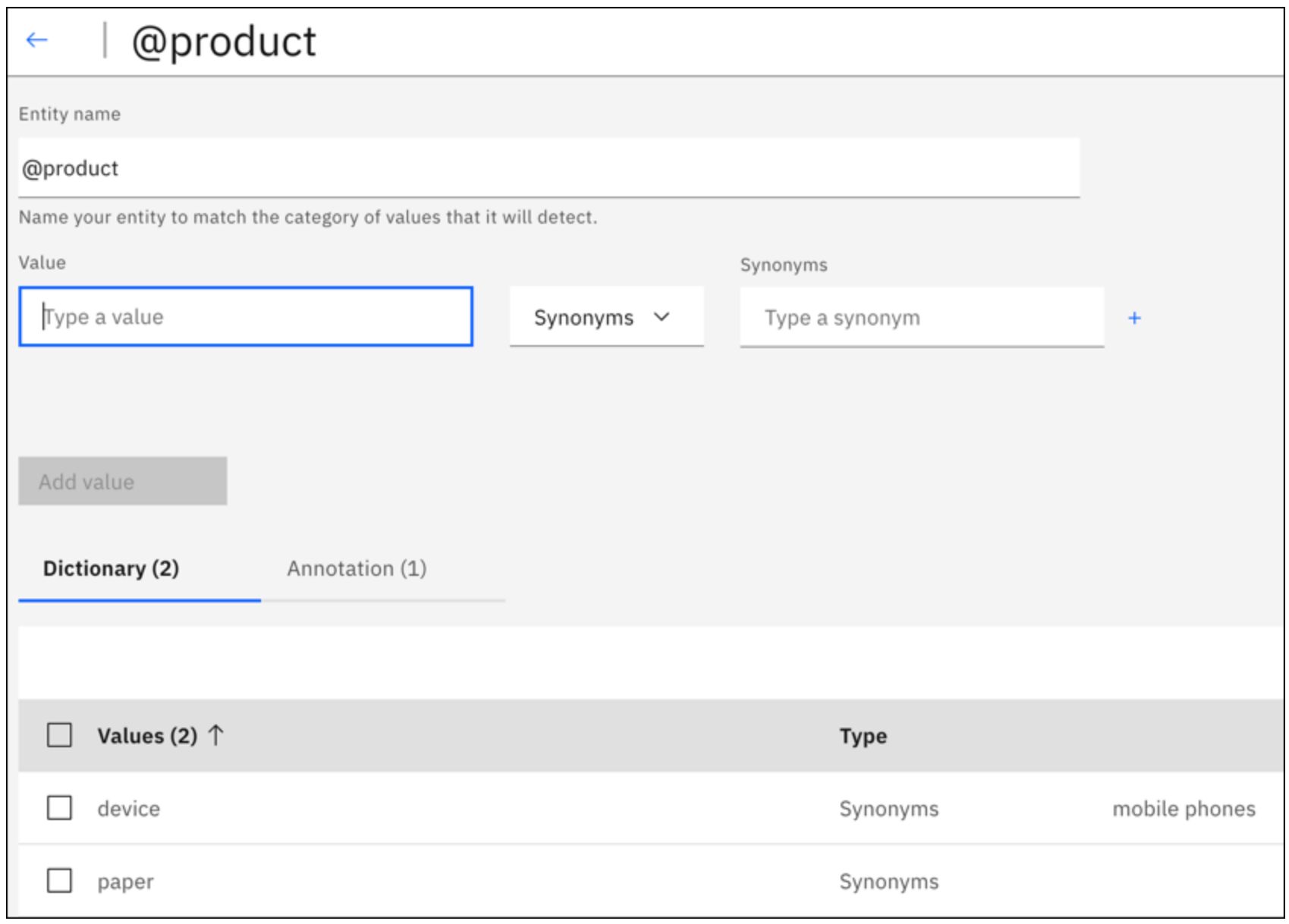
9. Click one of the annotations that you created. A box is displayed that says, `Go to: {entity-name}`. Clicking that link takes you directly to the entity.



Go to entity

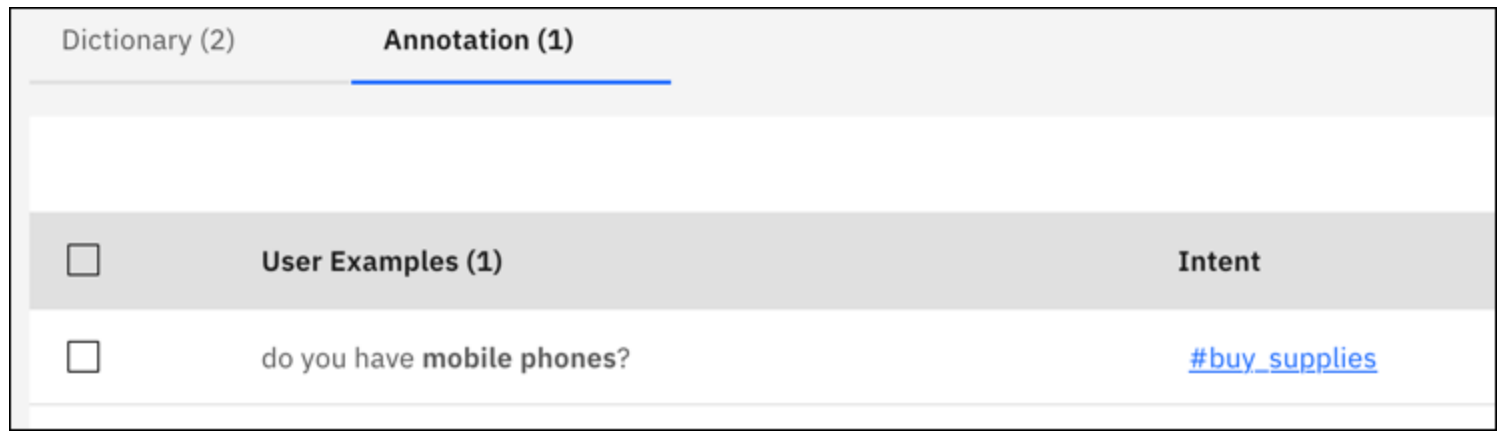
The annotation is added to the entity you associated it with, and the system trains itself on the new data.

⚠ **Important:** The term that you annotated is added to the entity as a new dictionary value. If you associated the annotated term with an existing entity value, then the term is added as a synonym of that entity value instead of as an independent entity value.



Mobile phones added as synonym

10. To see all of the mentions you annotated for a particular entity, from the entity's configuration page, click the **Annotation** tab.



See all annotations

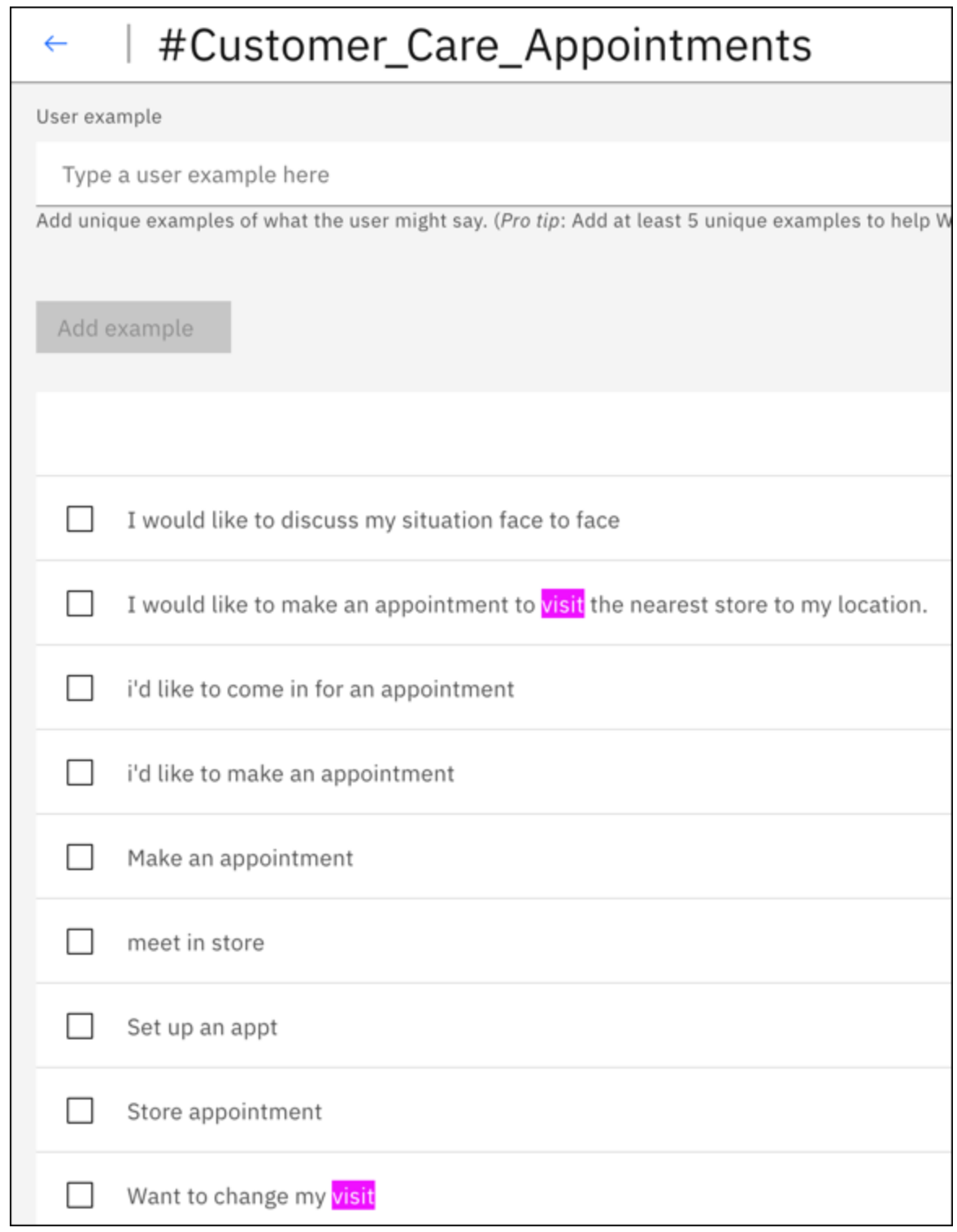
> **Note:** Contextual entities understand values that you didn't explicitly define. The system makes predictions about more entity values based on how your user examples are annotated, and uses those values to train other entities. Any similar user examples are added to the *Annotation* view, so you can see how this option impacts training.

If you do not want your contextual entities to use this expanded understanding of entity values, select all the user examples in the *Annotation* view for that entity, and then click **Delete**.

What you don't annotate matters

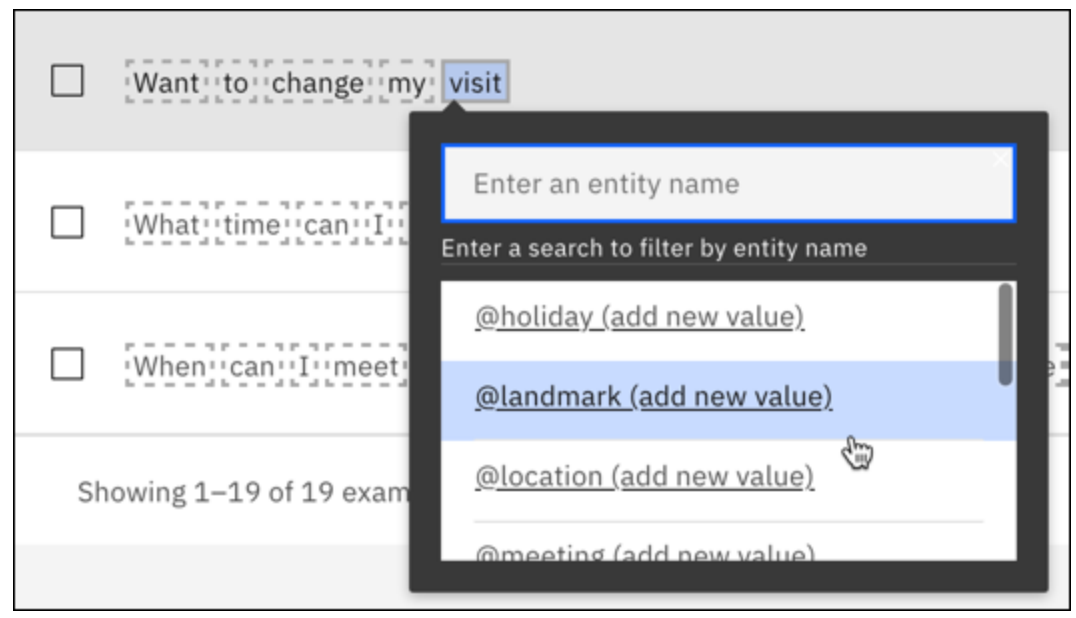
If you have an intent example with an annotation, and another word in that example matches the value or a synonym of the same entity, but the value is *not* annotated, that omission has impact. The model also learns from the context of the term that you did not annotate. Therefore, if you label one term as a mention of an entity in a user example, be sure to label any other applicable mentions also.

1. The `#Customer_Care_Appointments` intent includes two intent examples with the word `visit`.



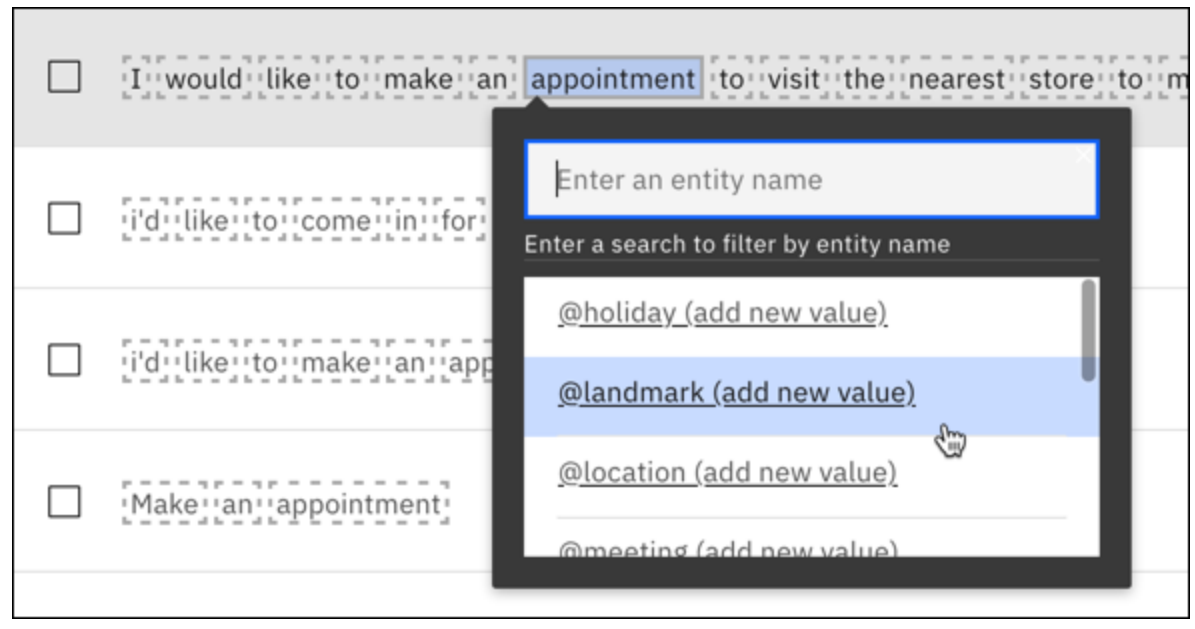
Examples with visit

2. In the second occurrence of the word, you want to annotate the word `visit` as an entity value of the `@meeting` entity. This annotation makes `visit` equivalent to other `@meeting` entity values such as `appointment`, as in *I'd like to make an appointment* or *I'd like to schedule a visit* .



Annotate visit with @meeting entity

3. In the first occurrence, the word `visit` is being used as a verb. It has a different meaning from a meeting. In this case, you can select the word `appointment` from the intent example, and annotate it as an entity value of the `@meeting` entity. The model learns from the fact that the word `visit` in the same example is not annotated.



Select appoinment

Enabling system entities

Watson Assistant provides a number of *system entities*, which are common entities that you can use for any application. Enabling a system entity makes it possible to quickly populate your skill with training data that is common to many use cases.

System entities can be used to recognize a broad range of values for the object types they represent. For example, the `@sys-number` system entity matches any numerical value, including whole numbers, decimal fractions, or even numbers written out as words.

System entities are centrally maintained, so any updates are available automatically. You cannot modify system entities.

1. On the Entities page, click **System entities**.
2. Browse through the list of system entities to choose the ones that are useful for your application.
   - To see more information about a system entity, including examples of matching input, click the entity in the list.
   - For details about the available system entities, see System entities.
3. Set the switch for each system entity that you want to use to **On**.

After you enable system entities, Watson Assistant retrains. After training is complete, you can use the entities.

Entity limits

The number of entities, entity values, and synonyms that you can create depends on your Watson Assistant service plan:

Plan	Entities per skill	Entity values per skill	Entity synonyms per skill
Enterprise	1,000	100,000	100,000
Premium (legacy)	1,000	100,000	100,000

Plus	1,000	100,000	100,000
Lite, Trial	100	100,000	100,000

Plan details

System entities that you enable for use count toward your plan usage totals.

Plan	Contextual entities and annotations
Enterprise	150 contextual entities with 3000 annotations
Premium (legacy)	150 contextual entities with 3000 annotations
Plus	100 contextual entities with 2000 annotations
Lite, Trial	10 contextual entities with 1000 annotations

Plan details continued

Editing entities

You can click any entity in the list to open it for editing. You can rename or delete entities, and you can add, edit, or delete values, synonyms, or patterns.

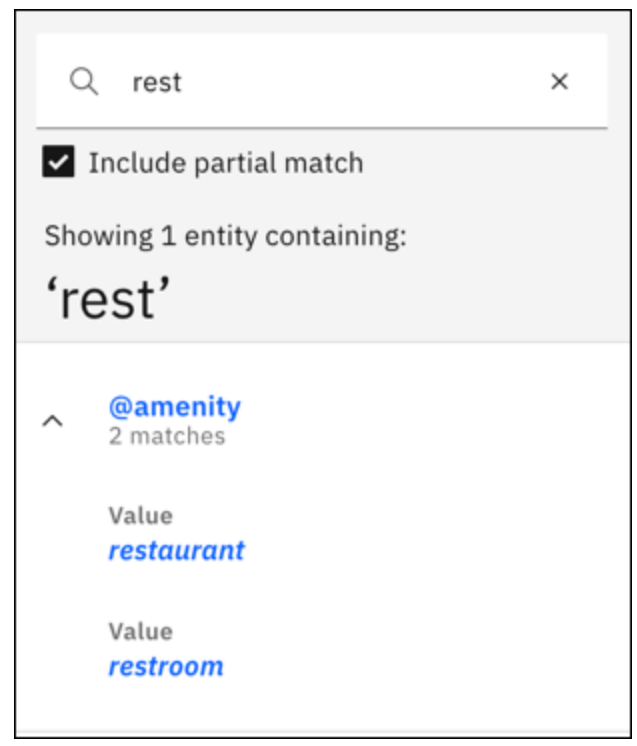
Note: If you change the entity type from `synonym` to `pattern`, or vice versa, the existing values are converted, but might not be useful as-is.

Searching entities

Use the Search feature to find entity names, values, and synonyms. System entities are not searchable.

1. From the **Entities** page, click the Search icon .
2. Enter a search term or phrase. You can also select **Include partial match**.

   The first time that you search for something, you might get a message that says the content is being indexed. If so, wait a minute, and then resubmit the search term.
3. Entities containing your search term, with corresponding examples, are shown.



Search results

Downloading entities

You can download a number of entities to a CSV file, so you can then upload and reuse them in another assistant.

- Pattern information is included in the downloaded CSV file. Any string that is wrapped with `/` is considered a pattern (as opposed to a synonym).
- Annotations that are associated with contextual entities are not downloaded. You must download the entire dialog to capture both the entity value and any associated annotations.

1. Go to the **Entities** page
    - To download all entities, do not select any individual entities. Instead, click the **Download all entities** icon ⭳.
    - To download the entities that are listed on the current page only, select the checkbox in the header. This action selects all of the entities on the current page. Then, click the **Download** button.
    - To download one or more specific entities, select the entities that you want to download, and then click the **Download** button.
2. Specify the name and location in which to store the CSV file that is generated, and then click **Save**.

Uploading entities

If you have many entities, you might find it easier to upload them from a comma-separated value (CSV) file than to define them one by one.

Note: Entity annotations are not included in the upload of an entity CSV file. You must upload the entire dialog to retain the associated annotations for a contextual entity. If you download and upload entities only, then any contextual entities that you downloaded are treated as dictionary-based entities after you upload them.

1. Collect the entities into a CSV file, or export them from a spreadsheet to a CSV file. The required format for each line in the file is as follows:

```
<entity>,<value>,<synonyms>
```

where <entity> is the name of an entity, <value> is a value for the entity, and <synonyms> is a comma-separated list of synonyms for that value.

```
weekday,Monday,Mon
weekday,Tuesday,Tue,Tues
weekday,Wednesday,Wed
weekday,Thursday,Thur,Thu,Thurs
weekday,Friday,Fri
weekday,Saturday,Sat
weekday,Sunday,Sun
month,January,Jan
month,February,Feb
month,March,Mar
month,April,Apr
month,May
```

Uploading a CSV file also supports patterns. Any string that is wrapped with `/` is considered a pattern (as opposed to a synonym).

```
ContactInfo,localPhone,/(\d{3})-(\d{4})/
ContactInfo,fullUSphone,/(\d{3})-(\d{3})-(\d{4})/
ContactInfo,internationalPhone,/^(\(?\+?[0-9]*\)?)?[0-9_\- \(\)]*$/
ContactInfo,email,/\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b/
ContactInfo,website,/(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
```

Tip: Save the CSV file with UTF-8 encoding and no byte order mark (BOM). The maximum CSV file size is 10 MB. If your CSV file is larger, consider splitting it into multiple files and uploading them separately. Open your dialog skill and then click the **Entities** tab.

2. Click the **Upload** icon ⭱.
3. Drag a file, or browse to select a file from your computer, and then click **Upload entities**.

   The file is validated and uploaded, and the system trains itself on the new data. You can view the uploaded entities on the Entities page.

Deleting entities

You can select a number of entities for deletion.

Important: When you delete an entity, you remove any values, synonyms, patterns, or annotations that are associated with the entity. This data cannot be retrieved later. All dialog nodes that reference these entities or values must be updated manually to no longer reference the deleted content.

1. Go to the **Entities** page.

- To delete all entities, do not select any individual entities. Instead, click the **Delete all entities** icon.
- To delete the entities that are listed on the current page only, select the checkbox in the header. This action selects all of the entities that are listed on the current page. Then, click the **Delete** button.
- To delete one or more specific entities, select the entities that you want to delete, and then click the **Delete** button.

Managing workflow with versions

Plus

Important: In the classic experience only, skill versions help you manage the workflow of a dialog skill development project. In watsonx Assistant, dialog can be included in your published versions of your content. For more information, see Publishing your content.

Create a skill version to capture a snapshot of the training data (intents and entities) and dialog in the skill at key points during the development process. Being able to save an in-progress skill at a specific point in time is especially useful as you start to fine-tune your assistant. You often need to make a change and see the impact before you can determine whether the change improves or lessens the effectiveness of the assistant. Based on your findings from a test environment deployment, you can make an informed decision about whether to deploy a change to an assistant that is deployed in a production environment.

Creating a version

You can edit only one version of the dialog skill at a time. The in-progress version is called the *development* version.

When you save a version, any skill settings that you applied to the development version are saved also.

To create a dialog skill version, follow these steps:

1. From the header of the skill, click **Save new version**, and then describe the current state of the skill.

   Adding a good description helps you to distinguish multiple versions from one another later.

   Tip: Add the date that you deploy the version to its description to make it easier to filter logs by version from the Analytics page later. For more information, see Picking a data source.

2. Click **Save**.

A snapshot is taken of the current skill and saved as a new version. You remain in the development version of the skill. Any changes that you make continue to be applied to the development version, not the version you saved. To access the version you saved, go to the **Versions** page.

If you have trouble, check that your skill does not have any entities with large numbers of values (such as 10,000 or more synonyms for a single entity). If it does, try to break the entity into many, more categorized entities, or consider using a contextual entity instead.

Deploying a skill version

1. Click **Versions**.
2. Click the ⋮ icon from the version that you want to deploy, and then choose **Assign**.

   A list of assistants to which you can link this version is displayed. The list is limited to those assistants without any skills, or that are associated with a different version of this skill.

3. Click the checkbox for one or more of the assistants, and then click **Assign**.

Important: Track when this version is deployed to an assistant and for how long. It is likely that you want to analyze user conversations that occur between users and this specific version of the skill. You can get this information from the **Analytics** page. However, when you pick a data source, versions are not listed. You must choose the name of the assistant to which you deployed this version. You can then filter the metrics data to show only those conversations that occurred between the start and end dates during which this skill version was deployed to the assistant.

Skill version limits

The number of versions you can create for a single skill depends on your plan.

Plan	Versions per skill

Enterprise	50
Premium (legacy)	50
Plus	10
Trial	10
Lite	0

Service plan details

Swapping skills

If the earlier version did a better job of recognizing and addressing customer needs than a later version, you can swap the skill to use the earlier version.

Follow the same steps that you use to deploy a skill version to change the version that is linked to an assistant.

Accessing a saved version

The only way to view a saved version is to overwrite the in-progress development version of the skill with the saved version. (But not before you save any work that you did in the current development version.)

You cannot edit a saved version. To achieve the same goal, you can use a saved version as the basis for a new version in which you incorporate any changes you want to make. To start development work from a saved version, overwrite the in-progress development version of the skill with the saved version.

1. Save any changes that you made to the skill since the last time you created a version.

   ⚠ **Important:** Save a version of the skill now. Otherwise, your work is lost when you follow these steps.

2. From the version you want to edit, click the **Click to view actions** ⋮ icon, and then choose **Revert** and confirm the action.

   The page refreshes to revert to the state that the skill was in when the version was created.

If you want to save any changes that you make to this version, you must save the skill as a new version. You cannot apply changes to an already-saved version.

⚠ **Important:** When you open a skill by clicking the skill tile from the assistant page, the development version of the skill is displayed. Even if you associated a later version with the assistant, when you access the skill, its development version opens.

Downloading a skill version

You can download a dialog skill version in JSON format. You might want to download a skill version if you want to use a specific version of a dialog skill in a different instance. You can download the version from one instance and import it to another instance as a new dialog skill.

To download a dialog skill version, complete the following steps:

1. From the skill menu, click **Versions**.
2. Click the **Click to view actions** ⋮ icon from the version that you want to download, and then choose **Download**.
3. Specify a name for the JSON file and where to save it, and then click **Save**.

Analytics (classic experience)

Dialog skill analytics overview

Classic experience only

This information applies to dialog skill analytics in the classic experience. For information about analytics in watsonx Assistant, see Use analytics to review your entire assistant at a glance.

The Analytics overview page provides a summary of the interactions between users and your assistant.

Analytics help you to understand the following things:

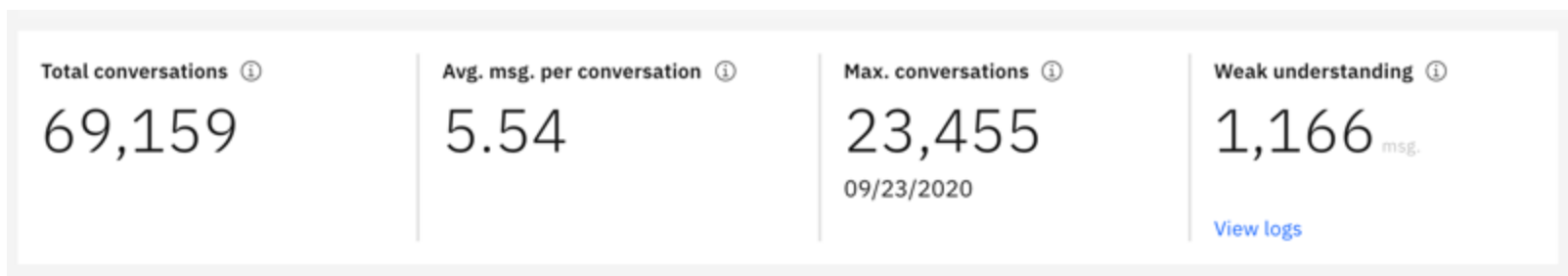
- *What do customers want help with today?*
- *Is your assistant understanding and addressing customer needs?*
- *How can you make your assistant better?*

To see metrics information, select **Overview** in the navigation bar. The information in Analytics is not immediately updated after a user interacts with your assistant. Watson Assistant prepares the data for Analytics in the background.

Note: After the numbers in metrics are greater than 3,000, the totals are approximated to prevent performance lags in the page.

Scorecards

The scorecards give you a quick view of your metrics. Scroll to see full interactive graphs later in the page.

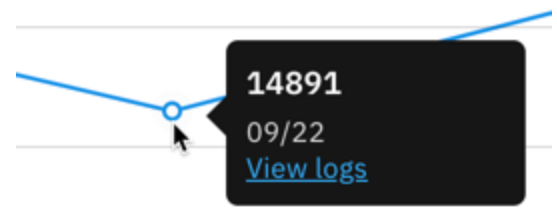


Scorecards

- *Total conversations*: The total number of conversations between active users and your assistant that occur during the selected time period. The total conversations metric and the *User conversations* page include only conversations in which both the assistant and customer participate. Conversations that have only welcome messages from the assistant, or that have only user messages of zero length, are not included. This metric is not used for billing purposes. An exchange with a user is not considered a billable conversation until the customer submits a message.
- *Avg. msg. per conversation*: The total messages that were received during the selected time period divided by the total conversations during the selected time period, as shown in the corresponding graph.
- *Max. conversations*: The maximum number of conversations for a single data point within the selected time period.
- *Weak understanding*: The number of individual messages with weak understanding. These messages are not classified by an intent, and do not contain any known entities. Reviewing unrecognized messages can help you to identify potential dialog problems.

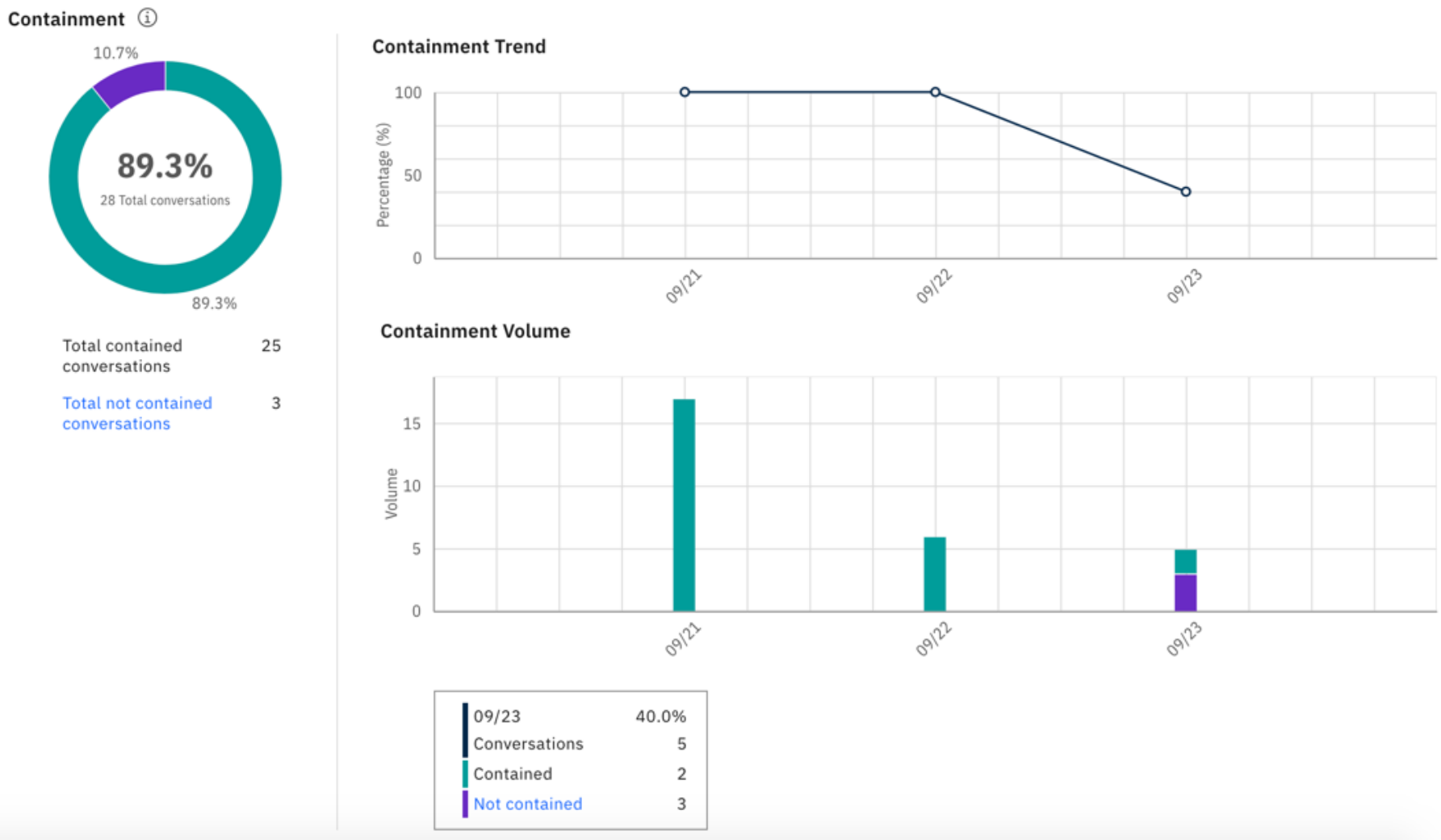
Graphs and statistics

Detailed graphs provide additional information. Click a data point on a graph to see more detail.



Data point

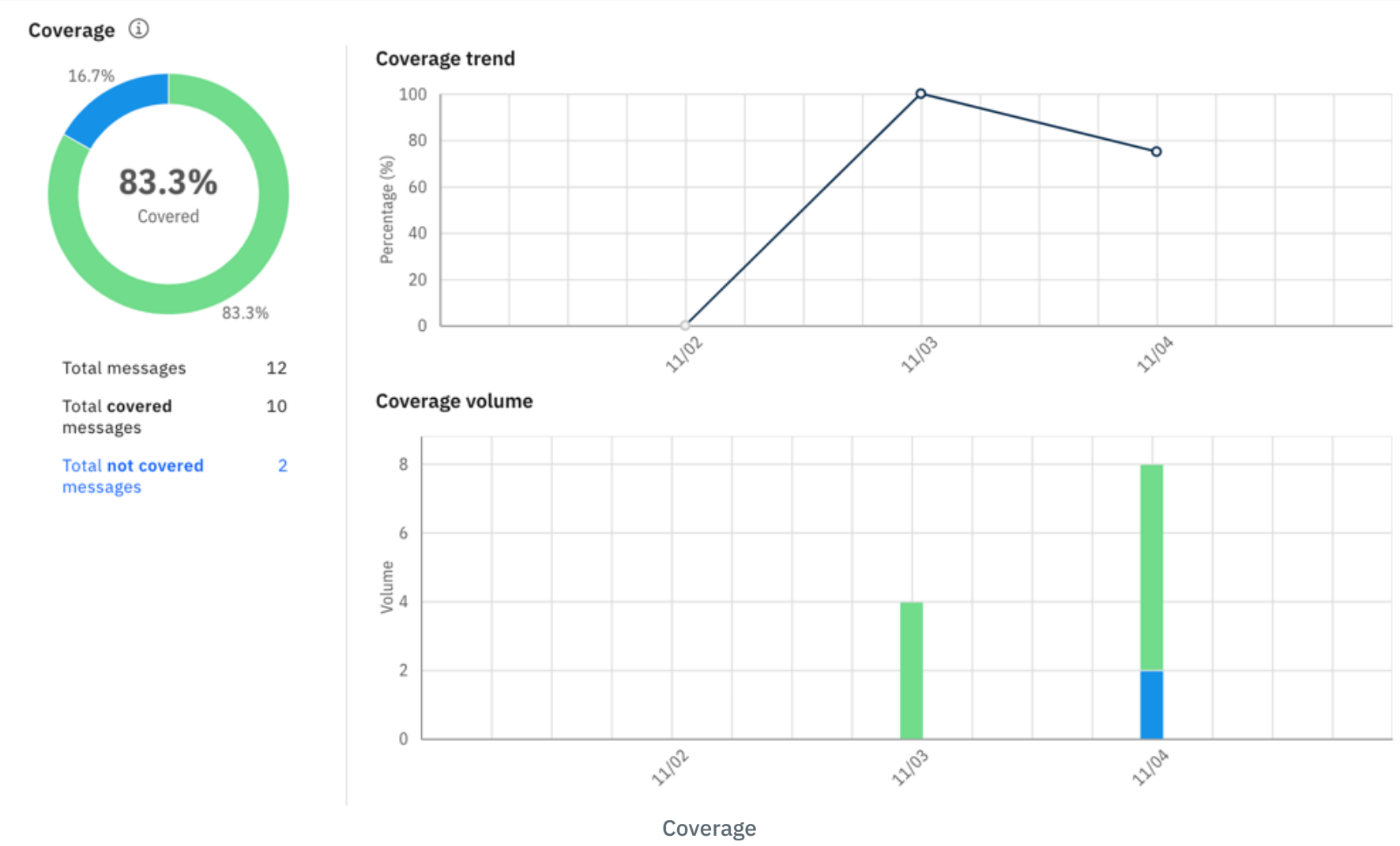
- *Containment*: Number of conversations in which the assistant is able to satisfy the customer's request without human intervention.
  - The volume graph shows the total number of conversations per day and how many of the conversations were contained and not contained.
  - The trend graph shows the percentage of daily conversations that were contained. This graph helps you to see if the assistant is getting better or worse at containing conversations over time.



Containment

The containment metric requires that your dialog flag requests for external support when they occur. For more information, see Measuring containment.

- *Coverage*: Number of conversations in which the assistant is confident that it can address a customer's request.
  - The volume graph shows the total number of conversations per day and how many of the conversations were covered (meaning intents in your dialog understood user requests and were able to address them), and not covered (meaning the input did not match an intent in the dialog and was processed by the *Anything else* node instead).
  - The trend graph shows the percentage of daily conversations that were covered. This graph helps you to see if your dialog is getting better or worse at covering conversations over time.



Coverage

The coverage metric requires that your dialog has an *Anything else* node. For more information, see Ending the conversation gracefully.

Note: The containment and coverage metrics are available to Plus or Enterprise plan users.

- *Total conversations*: The total number of conversations between active users and your assistant during the selected time period. This number counts

exchanges in which the welcome message is displayed to a user, even if the user doesn't respond.

- *Average messages per conversation* - The total messages that were received during the selected time period divided by the total conversations during the selected time period.
- *Total messages* - The total number of messages received from active users over the selected time period.
- *Active users* - The number of unique users who engaged with your assistant within the selected time period.
- *Average conversations per user* - The total conversations divided by the total number of unique users during the selected time period.

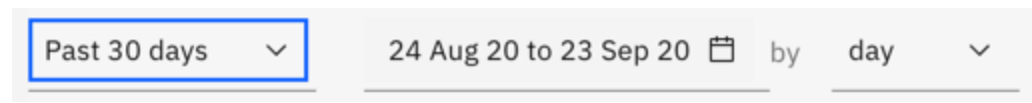
> ⚠ **Important:** Statistics for *Active users* and *Average conversations per user* require a unique `user_id` parameter to be specified with the messages. This value is typically specified by all integrations because it is used for billing purposes. For more information, see User-based plans explained.

Controls

You can use the following controls to filter the information:

- *Time period control* - Use this control to choose the period for which data is displayed. This control affects all data that is shown on the page: not just the number of conversations displayed in the graph, but also the statistics displayed along with the graph, and the lists of top intents and entities.

> **Note:** The statistics can cover a longer time period than the period for which logs of conversations are retained.

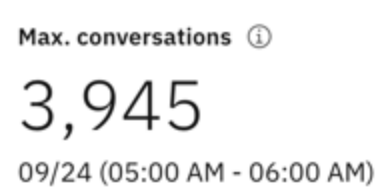


Time period control

You can choose whether to view data for a single day, a week, a month, or a quarter. In each case, the data points on the graph adjust to an appropriate measurement period. For example, when you view a graph for a day, the data is presented in hourly values, but when you view a graph for a week, the data is shown by day. A week always runs from Sunday through Saturday.

You can create custom time periods also, such as a week that runs from Thursday to the following Wednesday, or a month that begins on any date other than the first.

The time that is shown for each conversation reflects the time zone of your browser. However, API log calls are always shown in Coordinated Universal Time time. As a result, if you choose a single day view, for example, the time that is shown in the visualization might differ from the timestamp that is specified in the log for the same conversation.



Time period control

- *Intents* and *Entities* filters - Use either of these filters to show data for a specific intent or entity in your skill.

> ⚠ **Important:** The intent and entities filters are populated by the intents and entities in the skill, and not what is in the data source. If you selected a data source other than the skill, you might not see an intent or entity from your data source logs as an option in the filters, unless those intents and entities are also in the skill.

- *Refresh data*: Select **Refresh data** to refresh the data that is used in the page metrics.

  The statistics show traffic from customers who interact with your assistant; they do not include interactions from the *Try it out* pane.

Top intents and top entities

You can also view the intents and entities that were recognized most often during the specified time period.

- *Top intents* - Intents are shown in a simple list. You can see the number of times an intent was recognized. You can select an intent to open the **User conversations** page with a date range that is filtered to match the data you are viewing, and the intent that is filtered to match the selected intent.
- *Top entities* are also shown in a list. You can select an entity to open the **User conversations** page with the date range that is filtered to match the data you are viewing, and the entity that is filtered to match the selected entity.

See Improve your skill for tips on how to edit intents and entities based on discoveries you make by reviewing the intents and entities that your assistant

recognizes.

Improving your dialog skill

Classic experience only

This information applies to dialog skill analytics in the classic experience. For information about analytics in watsonx Assistant, see Use analytics to review your entire assistant at a glance.

The Analytics page provides a history of conversations between users and a deployed assistant. You can use this history to improve how your assistants understand and respond to user requests.

To open a list of individual messages between customers and the assistant that uses this dialog skill, select **User conversations** in the navigation bar.

When you open the **User conversations** page, the default view lists inputs that were submitted to the assistant for the last day, with the newest results first. The top intent (#intent) and any recognized entity (@entity) values used in a message, and the message text are available. For intents that are not recognized, the value that is shown is *Irrelevant*. If an entity is not recognized, or wasn't provided, the value that is shown is *No entities found*.

⚠ **Important:** The User conversations page displays the total number of *messages* between customers and your assistant. A message is a single utterance that a user sends to the assistant. A conversation typically consists of multiple messages. Therefore, the number of results on the **User conversations** page is different from the number of conversations that are shown on the **Overview** page.

Log limits

The length of time for which messages are retained depends on your service plan:

Plan	Message retention
Enterprise with Data Isolation	Last 90 days
Enterprise	Last 30 days
Premium (legacy)	Last 90 days
Plus	Last 30 days
Trial	Last 30 days
Lite	Last 7 days

Log limits

Filtering messages

You can filter messages by *Search user statements*, *Intents*, *Entities*, and *Last n days*.

Search user statements - Type a word in the search bar to search the users' inputs, but not your assistant's replies.

Intents - Select the menu and type an intent in the input field, or choose from the populated list. You can select more than one intent, which filters the results by using any of the selected intents, including *Irrelevant*.

Entities - Select the menu and type an entity name in the input field, or choose from the populated list. You can select more than one entity, which filters the results by any of the selected entities. If you filter by intent *and* entity, your results include the messages that have both values. You can also filter for results with *No entities found*.

Viewing individual messages

For any user input entry, click **Open conversation** to see the user input and the response that is made to it by the assistant within the context of the full conversation.

The time that is shown for each conversation reflects the time zone of your browser. This time might differ from the timestamp that is shown if you review the same conversation log with an API call; API log calls are always shown in Coordinated Universal Time.

You can then choose to show one or more classifications for the message you selected.

If the spell check feature is enabled for the skill, then any user utterances that were corrected are highlighted by the Auto-correct icon. The term that was corrected is underlined. You can hover over the underlined term to see the user's original input.

Improving across assistants

Creating a dialog skill is an iterative process. While you develop your skill, you can use the *Try it out* pane to verify that your assistant recognizes the correct intents and entities in test inputs, and to make corrections as needed.

From the User conversations page, you can analyze actual interactions between the assistant you used to deploy the skill and your users. Based on those interactions, you can make corrections to improve the accuracy with which intents and entities are recognized by your dialog skill. It is difficult to know exactly *how* your users ask questions, or what random messages they might submit, so it is important to frequently analyze real conversations to improve your dialog skills.

For an instance that includes multiple assistants, it might be useful to use message data from the dialog skill of one assistant to improve the dialog skill used by another assistant within that same instance.

As an example, you might have an instance that is named *HelpDesk*. You might have both a Production assistant and a Development assistant in your HelpDesk instance. When you work in the dialog skill for the Development assistant, you can use logs from the Production assistant messages to improve the Development assistant's dialog skill.

Any edits that you make within the dialog skill for the Development assistant affect only the Development assistant's dialog skill, even though you're using data from messages sent to the Production assistant.

Similarly, if you create multiple versions of a skill, you might want to use message data from one version to improve the training data of another version.

You cannot access log data from assistants that were created in other service instances.

Picking a data source

The term *data source* refers to the logs compiled from conversations between customers and the assistant or custom application by which a dialog skill was deployed.

When you open the *Analytics* page, metrics are shown that were generated by user interactions with the current dialog skill. No metrics are shown if the current skill isn't deployed for customers to use.

To populate the metrics with message data from a dialog skill or skill version that was added to a different assistant or custom application, one that interacted with customers, complete these steps:

1. Click the **Data source** field to see a list of assistants with log data that you might want to use.

   The list includes assistants that are deployed and to which you have access. Or you can choose to show a list of other deployments. For more information about other types of deployments, see *Show deployment IDs* explained.

2. Choose a data source.

Statistical information for the selected data source is displayed.

Notice that the list does not include skill versions. To get data that is associated with a specific skill version, you must know the time frame during which a specific skill version was used by a deployed assistant. You can select the assistant as the data source, and then filter the metrics by the appropriate dates to see log data that was generated by the assistant only while it was using the skill version.

Show deployment IDs explained

Applications that use the older v1 runtime API must specify a deployment ID in each message that is sent by using the `/message` API. This ID identifies the deployed app that the call was made from. The Analytics page can use this deployment ID to retrieve and display logs that are associated with a specific live application.

For assistants or custom apps that use the v2 version of the API, your assistant automatically includes an assistant ID with each `/message` call, so you can choose a data source by assistant name instead of using a deployment ID.

To add the deployment ID, v1 API users include the deployment property inside the metadata of the context, as in this example:

```
"context" : {
  "metadata" : {
      "deployment": "HelpDesk-Production"
  }
}
```

For an Enterprise with Data Isolation plan, you can ask IBM to configure your instances so you can access log data from deployed applications across

different instances. Each instance must use the v1 API and specify a deployment ID with each `/message` call. (If your instances use the v2 API, you cannot get log data from across different instances.)

> **Note:** If log sharing is enabled, only someone with Manager service access to the current instance can view logs. The person can view logs from all of the instances that are being shared. Logs from across all of the participating instances are displayed, regardless of the current user's service level access to the other instances. Similarly, when someone with Manager service access to an instance sends a GET request to the v1 `/logs` API endpoint, logs from all instances that participate in log sharing are returned, regardless of the user's service level access to each instance.

Making training data improvements

Use insights from real user conversations to correct the model associated with your dialog skill.

If you use data from another data source, any improvements you make to the model are applied to the current dialog skill only. The **Data source** field shows the source of the messages that you are using to improve this dialog skill, and the header of the page shows the dialog skill that you are applying changes to.

Correcting an intent

1. To correct an intent, click Edit icon.
2. From the list provided, select the correct intent for this input.
   - Begin typing in the entry field and the list of intents is filtered.
   - You can also choose **Mark as irrelevant** from this menu. (For more information, see Teaching your assistant about topics to ignore.) Or, you can choose **Do not train on intent**, which does not save this message as an example for training.
3. Select **Save**.

> **Tip:** The Watson Assistant service supports adding user input as an example to an intent *as-is*. If you are using @entity references as examples in your intent training data, and a user message that you want to save contains an entity value or synonym from your training data, then you must edit the message later. After you save it, edit the message from the Intents page to replace the entity that it references. For more information, see Directly referencing an @Entity as an intent example.

Adding an entity value or synonym

1. To add an entity value or synonym, click the Edit icon for the chosen entity.
2. Select **Add entity**.
3. Select a word or phrase in the underlined user input.
4. Choose an entity to which the highlighted phrase is added as a value.
   - Begin typing in the entry field and the list of entities and values is filtered.
   - To add the highlighted phrase as a synonym for an existing value, choose the `@entity:value` from the list.
5. Select **Save**.

Teaching your assistant about topics to ignore

It is important to help your assistant stay focused on the types of customer questions and business transactions that you designed it to handle. You can use information that is collected from real customer conversations to highlight subjects that you do not want your assistant to even attempt to address.

For more information, see Defining what's irrelevant.

Defining what's irrelevant

Classic experience only

This information applies to dialog skill analytics in the classic experience. For information about analytics in watsonx Assistant, see Use analytics to review your entire assistant at a glance.

Teach your dialog skill to recognize when a user asks about topics that it is not designed to answer.

To teach your assistant about subjects to ignore, you can review your user conversation logs to mark utterances that discuss off-topic subjects as irrelevant.

Intents that are marked as irrelevant are saved as counterexamples in the JSON workspace, and are included as part of the training data. They teach your assistant to explicitly not answer utterances of this type.

When you test your dialog, you can mark an intent as irrelevant directly from the *Try it out* pane.

Be certain before you designate an input as irrelevant.

- You can't access or change the inputs from the user interface later.
- The only way to reverse the identification of an input as being irrelevant is to use the same input in a test integration channel, and then explicitly assign it to an intent.

You might have subjects that you expect customers to ask and that you want your assistant to address eventually, but that you aren't ready to fully implement yet. Instead of adding those topics as counterexamples that can be hard to find later, capture the customer input examples as new intents. But don't link dialog nodes to the intents until you're ready. If customers ask about one of the topics, the anything_else node is triggered to explain that the assistant can't help with the current request, but can help them with other things.

Irrelevance detection

The *irrelevance detection* feature helps your dialog skill recognize subjects that you do not want it to address, even if you didn't teach it what to ignore. This feature helps your skill recognize irrelevant inputs earlier in the development process.

To test irrelevance detection in the "Try it out" pane, submit one or more utterances that have absolutely nothing to do with your training data. Irrelevance detection helps your skill to correctly recognize that the test utterances do not map to any of the intents defined in your training data, and classifies them as being `Irrelevant`.

The algorithmic models that help your assistant understand what your users say are built from two key pieces of information:

- Subjects that you want the assistant to address. For example, questions about order shipments for an assistant that manages product orders.

  You teach your assistant about these subjects by defining intents and providing sample user utterances that articulate the intents so your assistant can recognize these and similar requests as examples of input.

- Subjects that you want the assistant to ignore. For example, questions about politics for an assistant that makes pet grooming appointments exclusively.

  You teach your assistant about subjects to ignore by marking utterances that discuss subjects that are out of scope for your application as being irrelevant. Such utterances become counterexamples for the model.

The best way to build an assistant that understands your domain and the specific needs of your customers is to take the time to build good training data.

Counterexample limits

The maximum number of counterexamples that you can create for any plan type is 25,000.

Advanced analysis and log-related tasks

Classic experience only

This information applies to dialog skill analytics in the classic experience. For information about analytics in watsonx Assistant, see Use analytics to review your entire assistant at a glance.

Learn about notebooks and APIs that you can use to access and analyze log data.

Using Jupyter notebooks for analysis

IBM created Jupyter notebooks that you can use to analyze the behavior or your assistant. A Jupyter notebook is a web-based environment for interactive computing. You can run small pieces of code that process your data, and you can immediately view the results of your computation.

Note: You can use the notebooks with English-language skills only.

Analysis notebooks

Analysis notebooks are available for:

- IBM Watson® Studio
- Standard Python tools

Watson Studio provides an environment where you can:

- Choose the tools that you need to analyze and visualize data.
- Cleanse and shape data.
- Ingest streaming data.
- Create, train, and deploy machine learning models.

For more information, see the product documentation.

The watsonx Assistant Continuous Improvement Best Practices Guide describes how to get the most out of these notebooks.

Using the notebooks with Watson Studio

The following notebooks are available:

- Dialog skill analysis for watsonx Assistant
- Measure watsonx Assistant Performance.
- Analyze watsonx Assistant Effectiveness
- Dialog Flow Analysis for watsonx Assistant

If you choose to use the notebooks that are designed for use with Watson Studio, the steps are:

1. Create a Watson Studio account, create a project, and add a Cloud Object Storage account to it.
2. From the Watson Studio community, choose a notebook.

   Early in the development process, use the **Dialog skill analysis for watsonx Assistant** notebook to help you get started. The notebook:

   - Examines the terms that are correlated with each intent in your training data to find anomalies that might identify problems that you can investigate.
   - Uses a blind test set that you provide to calculate performance on statistical metrics like Accuracy, Precision, Recall, and F1.
   - Offers advanced features that you can use to find the causes of common issues, such as why some sentences are often misidentified.

   To learn more about how this notebook can help you improve your dialog, read Dialog skill analysis.
3. After you deploy a version of the assistant and collect conversation log data, run the **Measure watsonx Assistant Performance** notebook.
4. Follow the step-by-step instructions provided with the notebook to analyze a subset of the dialog exchanges from the logs.

   Run the following notebook first:

   - **Measure**: Gathers metrics that focus on coverage (how often the assistant is confident enough to respond to users) and effectiveness (when the assistant does respond, whether the responses are satisfying user needs).

   The insights are visualized in ways that make it easier to understand areas for improvement in your assistant.
5. Export a sample set of the logs from ineffective conversations, and then analyze and annotate them.

   For example, indicate whether a response is correct. If correct, mark whether it is helpful. If a response is incorrect, then identify the root cause, the wrong intent or entity was detected, for example, or the wrong dialog node was triggered. After you identify the root cause, indicate what is the correct choice.
6. Feed the annotated spreadsheet to the **Analyze watsonx Assistant Effectiveness** notebook.

   - **Effectiveness**: Provides a deeper analysis of your logs to help you understand the steps that you can take to improve your assistant.
7. Use the **Dialog Flow Analysis for watsonx Assistant** notebook to review your dialog. The notebook can help you pinpoint the dialog nodes where customers most frequently abandon the conversation.

   For more information about how this notebook can help you analyze and assess abandonment, see Do you know where and why users drop off the conversation?.

This process helps you to understand the steps you can take to improve your assistant.

Using the notebooks with standard Python tools

If you choose to use standard Python tools to run the notebooks, you can get the notebooks from GitHub.

- Dialog Skill Analysis for watsonx Assistant
- IBM® watsonx™ Assistant Recommendation notebooks (Measure and Analyze Effectiveness)
- IBM® watsonx™ Assistant Dialog Flow Analysis notebook

The watsonx Assistant Continuous Improvement Best Practices Guide outlines which notebook to use at each stage of your improvement process.

Using the logs API

You can use the `/logs` API to list events from the transcripts of conversations that occurred between your users and your assistant. For conversations created with the v2 `/message` API, use the instance-level endpoint to list log events in all workspaces, and then filter by Assistant ID. For more information, see Filter query reference.

> ⚠ **Important:** The API logs messages that are exchanged in conversations that are defined by a dialog skill only.

The number of days that logs are stored differs by service plan type. For more information, see Log limits.

For a Python script you can run to export logs and convert them to CSV format, download the `export_logs_py.py` file from the watsonx Assistant GitHub repository).

Understanding terminology

First, review the definitions of terms that are associated with Watson Assistant logs.

Term	Definition
Assistant	An application - sometimes referred to as a 'chat bot' - that implements your Watson Assistant content.
Assistant ID	The unique identifier of an assistant.
Conversation	A set of messages that an individual user sends to your assistant, and the messages your assistant sends back.
Conversation ID	Unique identifier that is added to individual message calls to link related message exchanges together. App developers that use the V1 version of the Watson Assistant API add this value to the message calls in a conversation by including the ID in the metadata of the context object.
Customer ID	A unique ID that can be used to label customer data such that it can be deleted if the customer requests the removal of their data.
Deployment ID	A unique label that app developers of the Watson Assistant API V1 version pass with each user message to help identify the deployment environment that produced the message.
Instance	Your deployment of Watson Assistant, accessible with unique credentials. A Watson Assistant instance might contain multiple assistants.
Message	A message is a single utterance that a user sends to the assistant.
Skill ID	The unique identifier of a skill.
User	A user is anyone who interacts with your assistant.
User ID	A unique label that is used to track the level of service usage of a specific user.

Terminology

> ⚠ **Important:** The **User ID** property is not equivalent to the **Customer ID** property, though both can be passed with the message. The **User ID** field is used to track levels of usage for billing purposes. The **Customer ID** field is used to support the labeling and subsequent deletion of messages that are associated with users. Customer ID is used consistently across all Watson services and is specified in the `X-Watson-Metadata` header. User ID is used exclusively by the Watson Assistant service and is passed in the context object of each `/message` API call.

Associating message data with a user for deletion

There may be cases when you want to completely remove a set of your user's data from a Watson Assistant instance. When the delete feature is used, the Overview metrics don't include those deleted messages, and results in fewer Total Conversations.

Before you begin

To delete messages for one or more individuals, you first need to associate a message with a unique **Customer ID** for each individual. To specify the

Customer ID for any message that is sent with the `/message` API, include the `X-Watson-Metadata: customer_id` property in your header. You can pass multiple **Customer ID** entries with semicolon separated `field=value` pairs, by using `customer_id`, as in the following example:

```
$ curl -X POST -u "apikey:3Df... ...Y7Pc9" \
 --header \
   "Content-Type: application/json" \
   "X-Watson-Metadata: customer_id={first-customer-ID};customer_id={second-customer-ID}" \
 --data "{\"input\":{\"text\":\"hello\"}}" \
 "{url}/v2/assistants/{assistant_id}/sessions/{session_id}/message?version=2019-02-28"
```

where {url} is the appropriate URL for your instance. For more information, see Endpoint URLs.

Note: The `customer_id` string cannot include the semicolon (`;`) or equal sign (`=`) characters. You are responsible for ensuring that each `Customer ID` parameter is unique across your customers.

For instructions on how to delete messages that use `customer_id` values, see the Labeling and deleting data in watsonx Assistant.

Dialog tutorials

Getting started with dialog

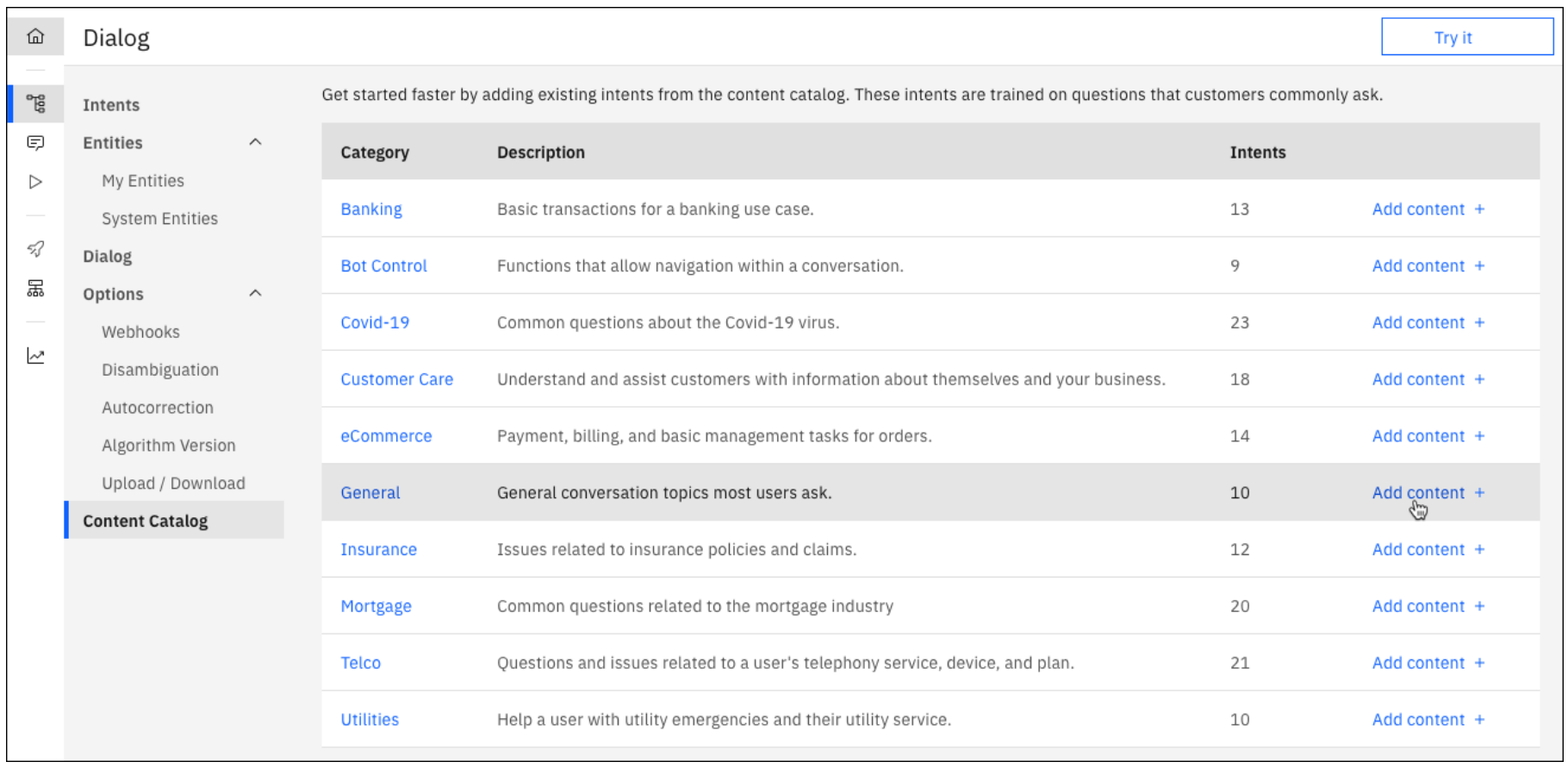
In this short tutorial, we help you use a dialog to build your first conversation.

A *dialog* uses natural language processing and machine learning technologies to understand user questions and requests, and respond to them with answers that are authored by you.

Step 1: Add intents from a content catalog

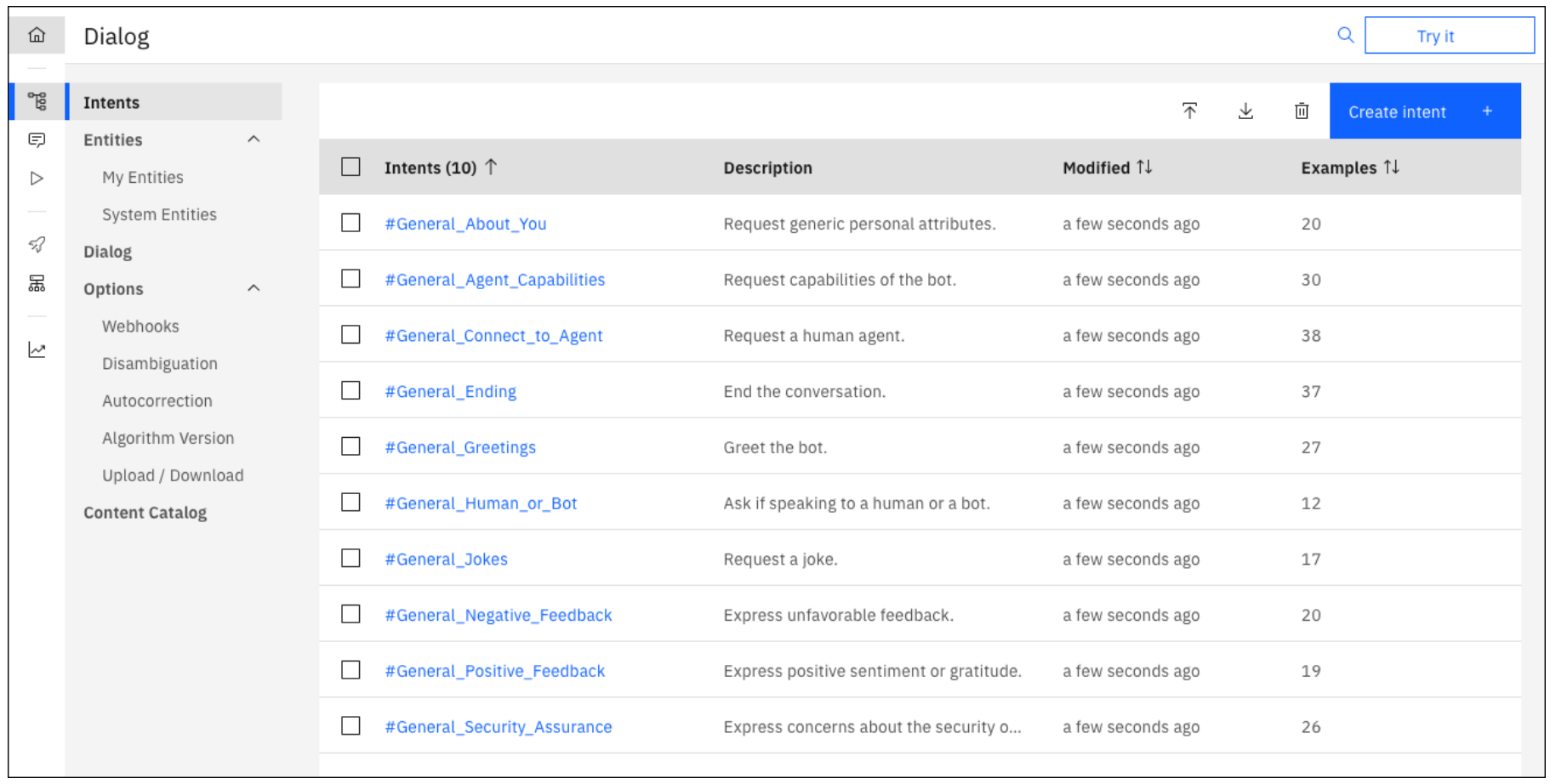
The Intents page is where you start to train your assistant. In this tutorial, you add training data that was built by IBM to your skill. Prebuilt intents are available from the content catalog. You give your assistant access to the **General** content catalog so your dialog can greet users and end conversations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

1. Click **Content Catalog**.
2. Find **General** in the list, and then click **Add content**.



Content Catalog

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



Intents

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

Step 2: Build a dialog

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

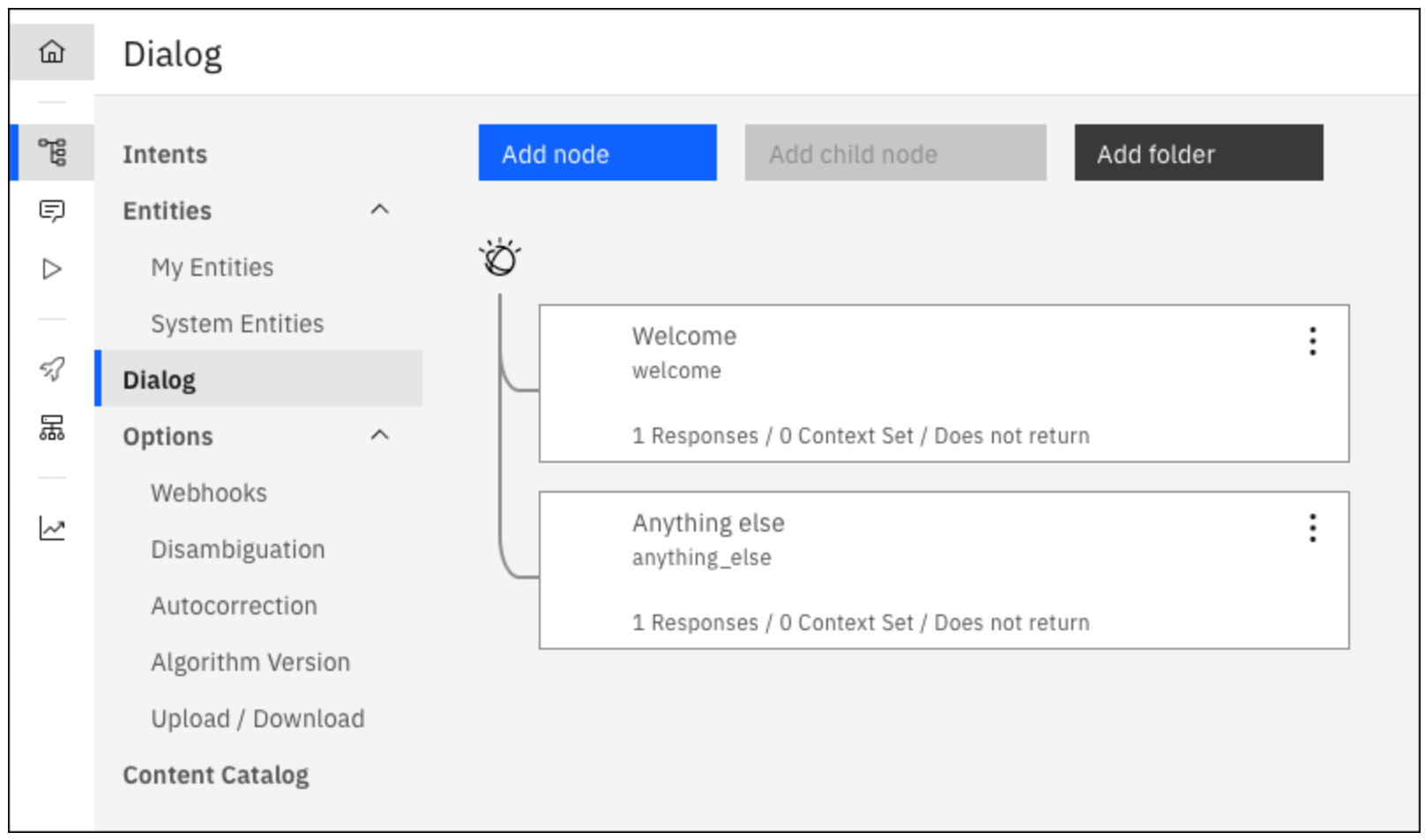
We'll create a simple dialog that handles greeting and ending intents, each with a single node.

Adding a start node

1. Click **Dialog**.

   The following two dialog nodes are created for you automatically:

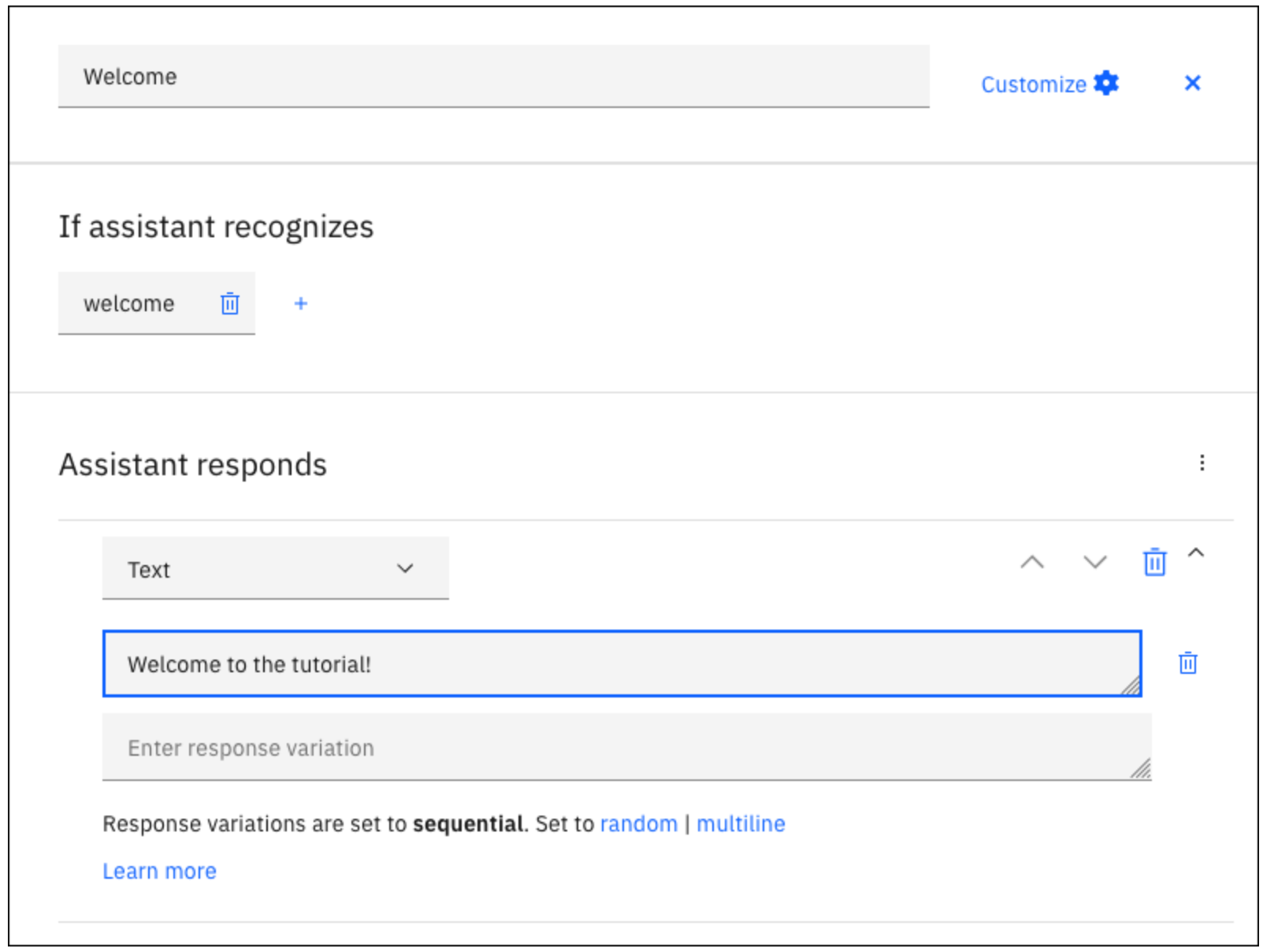
   - **Welcome**: Contains a greeting that is displayed to your users when they first engage with the assistant.
   - **Anything else**: Contains phrases that are used to reply to users when their input is not recognized.



New dialog with built-in nodes

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

3. Replace the default response with the text `Welcome to the tutorial!`.



Welcome node

4. Click to close the edit view.

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

Testing the start node

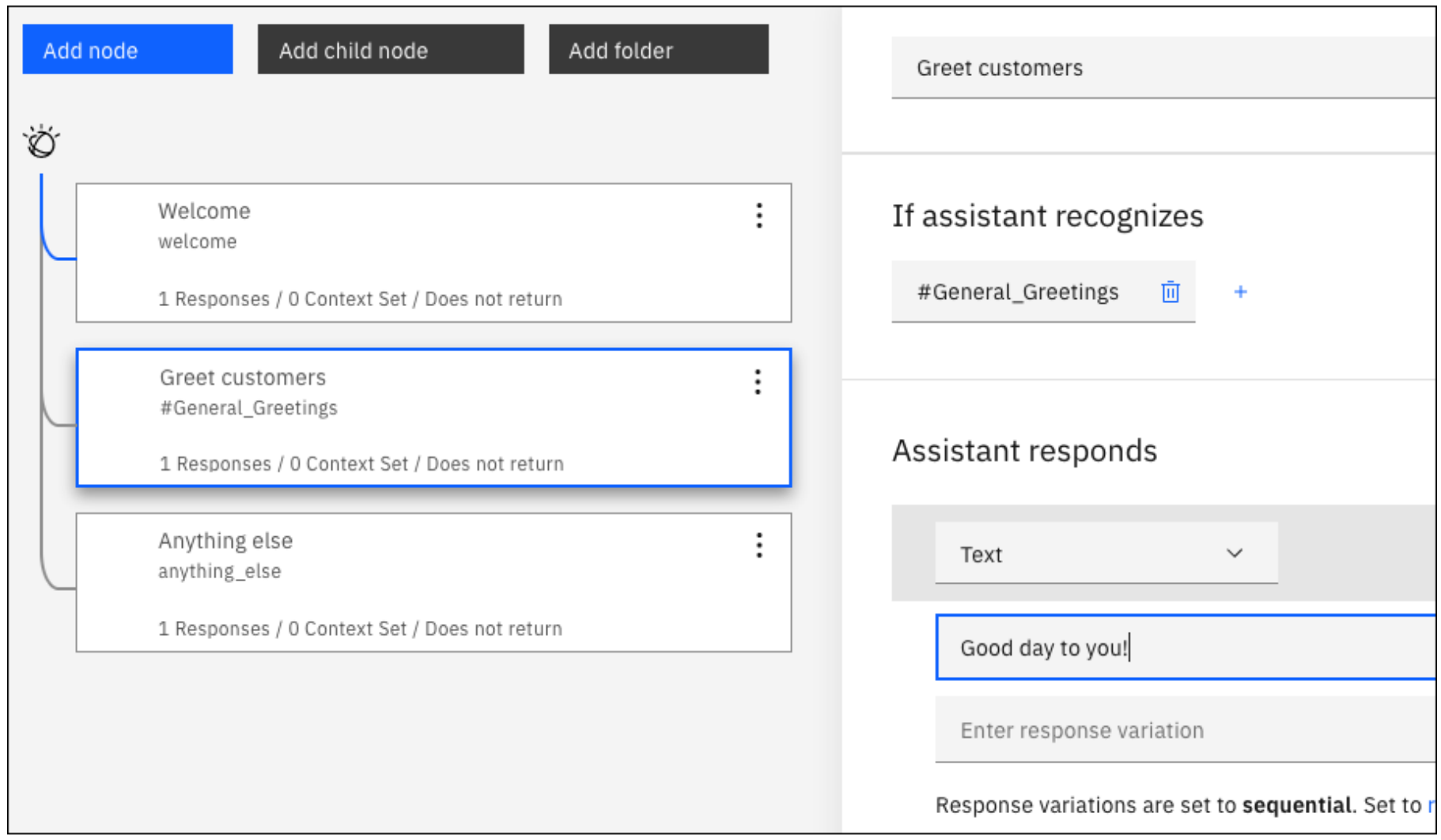
You can test your dialog at any time to verify the dialog.

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

Adding nodes to handle intents

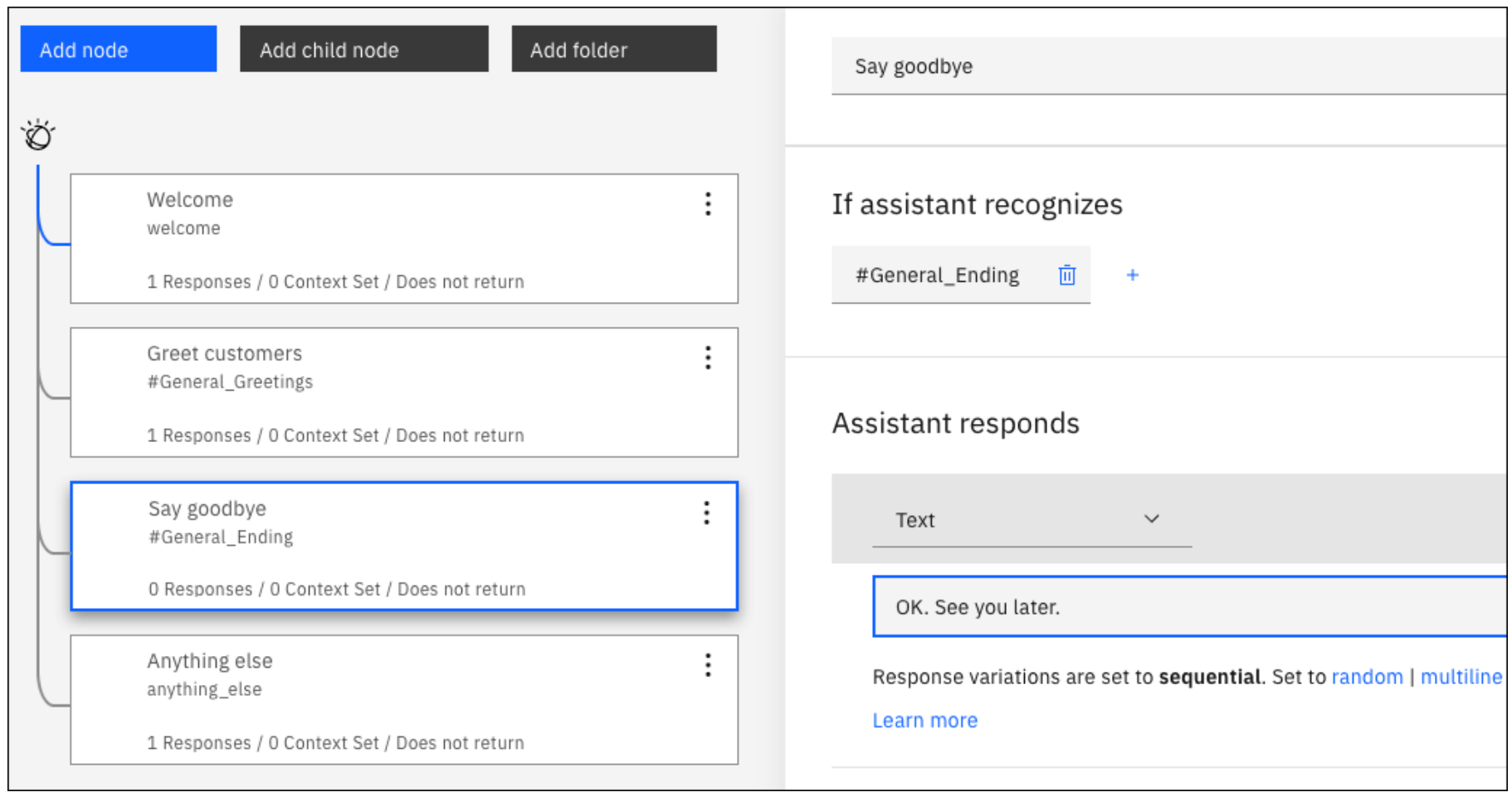
Now add nodes between the `Welcome` node and the `Anything else` node that handle our intents.

1. Click **Add node**.
2. In the node name field, type `Greet customers`.
3. In the **If assistant recognizes** field of this node, start to type `#General_Greetings`. Then, select the `#General_Greetings` option.
4. Add the response text, `Good day to you!`



Greeting node

5. Click to close the edit view.
6. Click **Add node** to create a peer node.
7. Name the peer node `Say goodbye` and specify `#General_Ending` in the **If assistant recognizes** field.
8. Add `OK. See you later.` as the response text.



Ending node

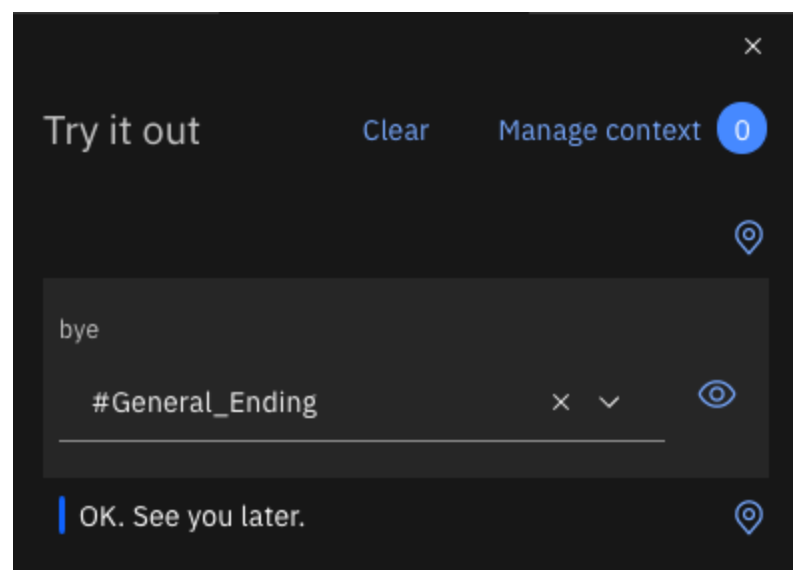
9. Click to close the edit view.

Testing intent recognition

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

1. Click **Try it** to open the "Try it out" pane.
2. In the text field, type `Hello` and then press Enter. The output indicates that the `#General_Greetings` intent was recognized, and the appropriate response (`Good day to you.`) is displayed.
3. Try the following input:

- bye
- howdy
- see ya
- good morning
- sayonara



Testing in Try it out

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

Result of building a dialog

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

Next steps

This tutorial is built around a simple example. For a real application, you need to define some more interesting intents, some entities, and a more complex dialog that uses them both. When you have a polished version of the assistant, you can integrate it with websites or channels, such as Slack, that your customers already use. As traffic increases between the assistant and your customers, you can use the tools that are provided in the **Analytics** page to analyze real conversations, and identify areas for improvement.

Building a complex dialog

In this tutorial, you create a dialog for an assistant that helps users with inquiries about a fictitious restaurant called *Truck Stop Gourmand*.

Learning objectives

By the time you finish the tutorial, you learn how to:

- Plan a dialog.
- Define custom intents.
- Add dialog nodes that can handle your intents.
- Add entities to make your responses more specific.
- Add a pattern entity, and use it in the dialog to find patterns in user input.
- Set and reference context variables.

Duration

This tutorial takes approximately 2 to 3 hours to complete.

Prerequisite

Before you begin, complete the Getting started with dialog tutorial.

You use the dialog that you created, and add nodes to the simple dialog that you built as part of the getting started exercise.

Step 1: Plan the dialog

You are building an assistant for a restaurant that is named *Truck Stop Gourmand* that has one location and a thriving cake-baking business. You want the simple assistant to answer user questions about the restaurant, its menu, and to cancel customer cake orders. Therefore, you need to create intents that handle inquiries related to the following subjects:

- Restaurant information
- Menu details
- Order cancellations

You start by creating intents that represent these subjects, and then build a dialog that responds to user questions about them.

Step 2: Answer questions about the restaurant

Add an intent that recognizes when customers ask for details about the restaurant itself. An intent is the purpose or goal that is expressed in user input. The `#General_About_You` intent that is provided with the *General* content catalog serves a similar function, but its user examples are designed to focus on queries about the assistant as opposed to the business that is using the assistant to help its customers. So, you add your own intent.

Add the #about_restaurant intent

1. On the **Intents** page, click **Create intent**.
2. Add the following intent name, and then click **Create intent**:

   ```
   about_restaurant
   ```

   The #about_restaurant intent is added. A number sign (`#`) prefix is added to the intent name to label it as an intent. This naming convention helps you and others recognize the intent as an intent. It has no example user utterances that are associated with it yet.
3. In the **Add user examples** field, enter the following utterances:

   ```
   Tell me about the restaurant
   i want to know about you
   who are the restaurant owners and what is their philosophy?
   What's your story?
   Where do you source your produce from?
   Who is your head chef and what is the chef's background?
   How many locations do you have?
   do you cater or host functions on site?
   Do you deliver?
   Are you open for breakfast?
   ```
4. Click the **Close** arrow ← to finish adding the `#reservation` intent and its example utterances.

You added an intent and provided examples of utterances that real users might enter to trigger this intent.

Add a dialog node that is triggered by the #about_restaurant intent

Add a dialog node that recognizes when the user input maps to the intent that you created in the previous step, meaning its condition checks whether your assistant recognized the `#about_restaurant` intent from the user input.

1. Click **Dialog** to open the dialog tree.
2. Find the **#General_Greetings** node in the dialog tree.

   You will add a node that checks for questions about the restaurant after this initial greeting node to reflect the flow you might expect to encounter in a normal conversation. For example, `Hello.` then `Tell me about yourself.`
3. Click the Node options icon ⋮ on the **Greet customers** node, and then select **Add node below**.
4. Start typing `#about_restaurant` in the **If assistant recognizes** field, and then select it from the list.

   This node is used if the user input matches the `#about_restaurant` intent.
5. In **Assistant responds**, enter the text response:

   ```
   Truck Stop Gourmand is the brainchild of Gloria and Fred Smith. What started out as a food truck in 2004 has expanded into a thriving restaurant. We now have one brick-and-mortar restaurant in downtown Portland. The bigger kitchen brought with it new chefs, but each one is faithful to the philosophy that made the Smith food truck so popular to begin with: deliver fresh, local produce in inventive and delicious ways. Join us for lunch or dinner seven days a week. Or order a cake from our bakery.
   ```

Add an image to the response also.

1. Click **Add response type**.
2. Select **Image** from the drop-down list. In the **Image source** field, add `https://www.ibmlearningcenter.com/wp-content/uploads/2018/02/IBM-Learning-Center-`

Food4.jpg .

3. Move up the image response type, so it is displayed in the response before the text is displayed. Click the **Move up** arrow to reorder the two response types.

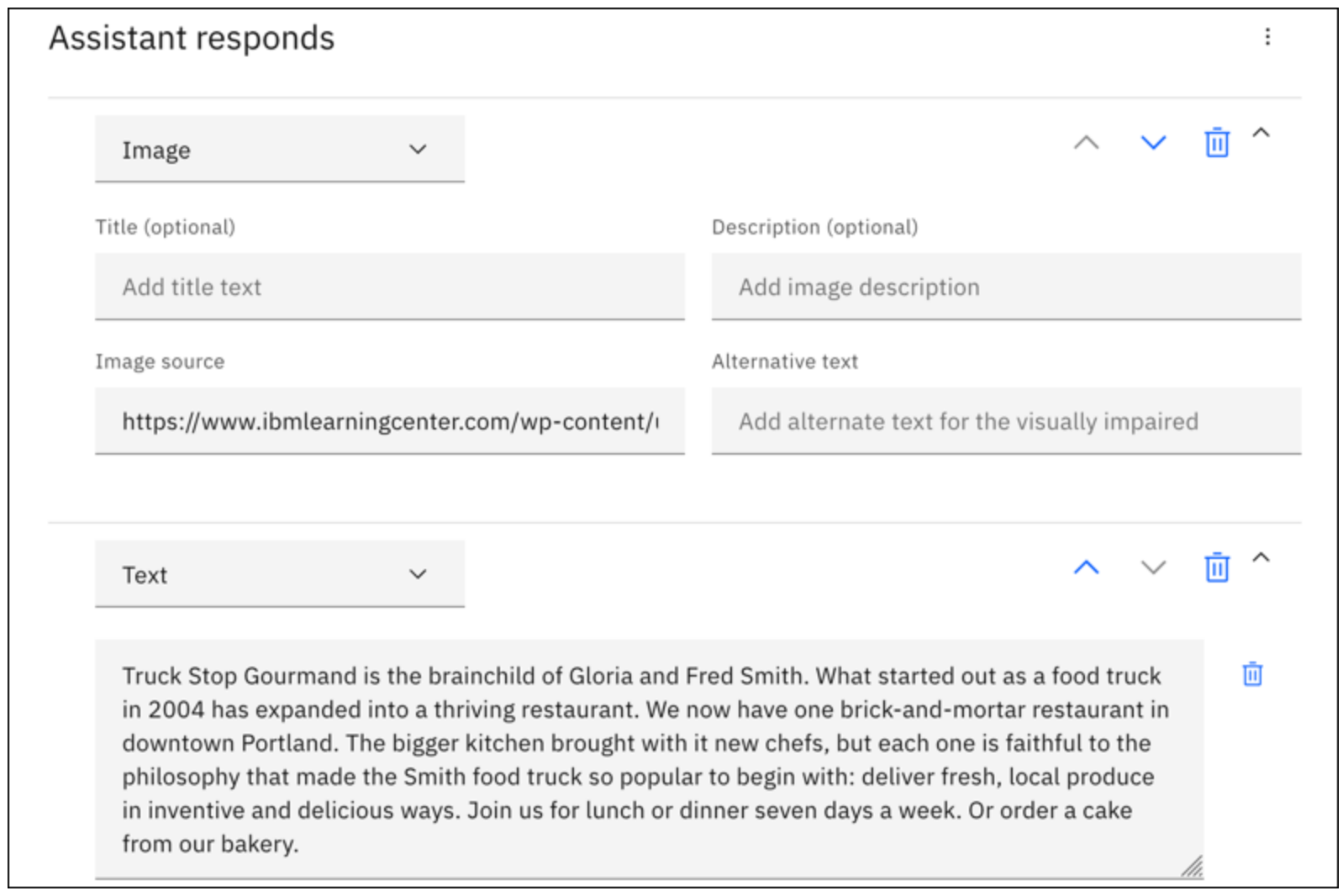


Image before text response and response

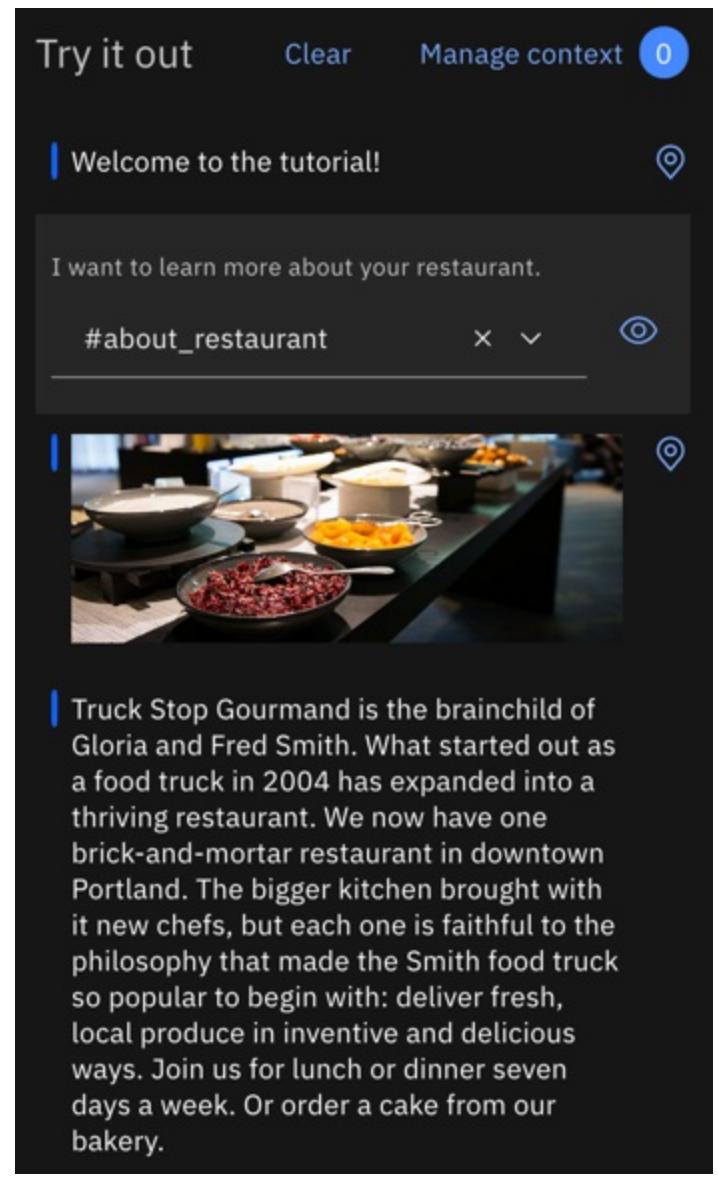
4. Click the close icon to close the node edit view.

Test the #about_restaurant dialog node

Test the intent by checking whether user utterances that are similar to, but not the same as, the examples you added have trained your assistant to recognize input with an `#about_restaurant` intent.

1. Click **Try it**.
2. Type `I want to learn more about your restaurant.`

   Your assistant indicates that the `#about_restaurant` intent is recognized, and returns a response with the image and text that you specified for the dialog node.



Try it out

You added a custom intent, and a dialog node that knows how to handle it.

The `#about_restaurant` intent is designed to recognize various general questions about the restaurant. You added a single node to capture such questions. The response is long, but it is a single statement that can potentially answer questions about all of the following topics:

- Restaurant owners
- Restaurant history
- Philosophy
- Number of sites
- Days of operation
- Meals served
- The restaurant bakes cakes to order

For general questions, a single, general answer is suitable.

Step 3: Answer questions about the menu

A key question from potential restaurant customers is about the menu. The Truck Stop Gourmand restaurant changes the menu daily. In addition to its standard menu, it has vegetarian and cake shop menus. When a user asks about the menu, the dialog needs to find out which menu to share, and then provide a link to the menu on the restaurant's website. You never want to hardcode information into a dialog node if that information changes regularly.

Add a #menu intent

1. On the **Intents** page, click **Create intent**.
2. Add the following intent name, and then click **Create intent**:

   ```
   menu
   ```

3. In the **Add user examples** field, enter the following utterances:

   ```
   I want to see a menu
   What do you have for food?
   Are there any specials today?
   where can i find out about your cuisine?
   What dishes do you have?
   What are the choices for appetizers?
   do you serve desserts?
   ```

```
What is the price range of your meals?
How much does a typical dish cost?
tell me the entree choices
Do you offer a prix fixe option?
```

4. Click the **Close** arrow ← icon to finish adding the `#menu` intent and its example utterances.

Add a dialog node that is triggered by the #menu intent

Add a dialog node that recognizes when the user input maps to the intent that you created in the previous step, meaning its condition checks whether your assistant recognized the `#menu` intent from the user input.

1. Click **Dialog** to open the dialog tree.
2. Find the **#about_restaurant** node in the dialog tree.

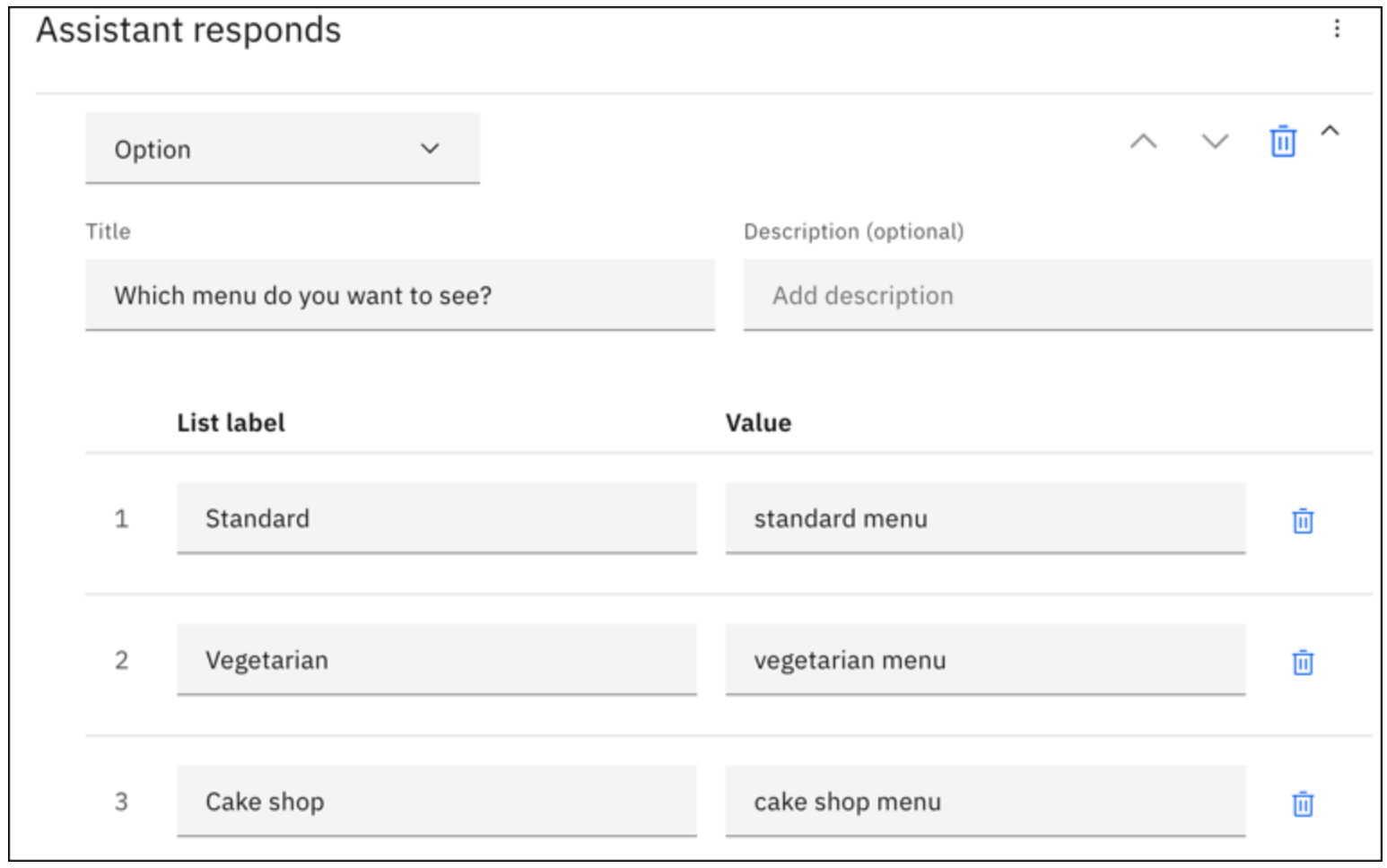
   You will add a node that checks for questions about the menu after this node.
3. Click the Node options icon ⋮ on the **#about_restaurant** node, and then select **Add node below**.
4. Start typing `#menu` in the **If assistant recognizes** field, and then select it from the list.

   This node is used if the user input matches the `#menu` intent.
5. In **Assistant responds**, enter the text response:

   ```
   In keeping with our commitment to giving you only fresh local ingredients, our menu changes daily to accommodate the produce we pick up in the morning. You can find today's menu on our website.
   ```
6. Add an *option* response type that provides a list of options for the user to choose from. In this case, the list of options includes the different versions of the menu that are available. Click **Add response type**.
7. Select **Option** from the drop-down list.
8. In the **Title** field, add `Which menu do you want to see?`.
9. Click **Add option**.
10. In the **List label** field, add `Standard`. The text that you add as the label is displayed in the response to the user as a selectable option.
11. In the **Value** field, add `standard menu`. The text that you specify as the value is what gets sent to your assistant as new user input when a user chooses this option from the list, and clicks it.
12. Repeat the previous two steps to add label and value information for the remaining menu types:

List label	Value
Vegetarian	vegetarian menu
Cake shop	cake shop menu

Menu options



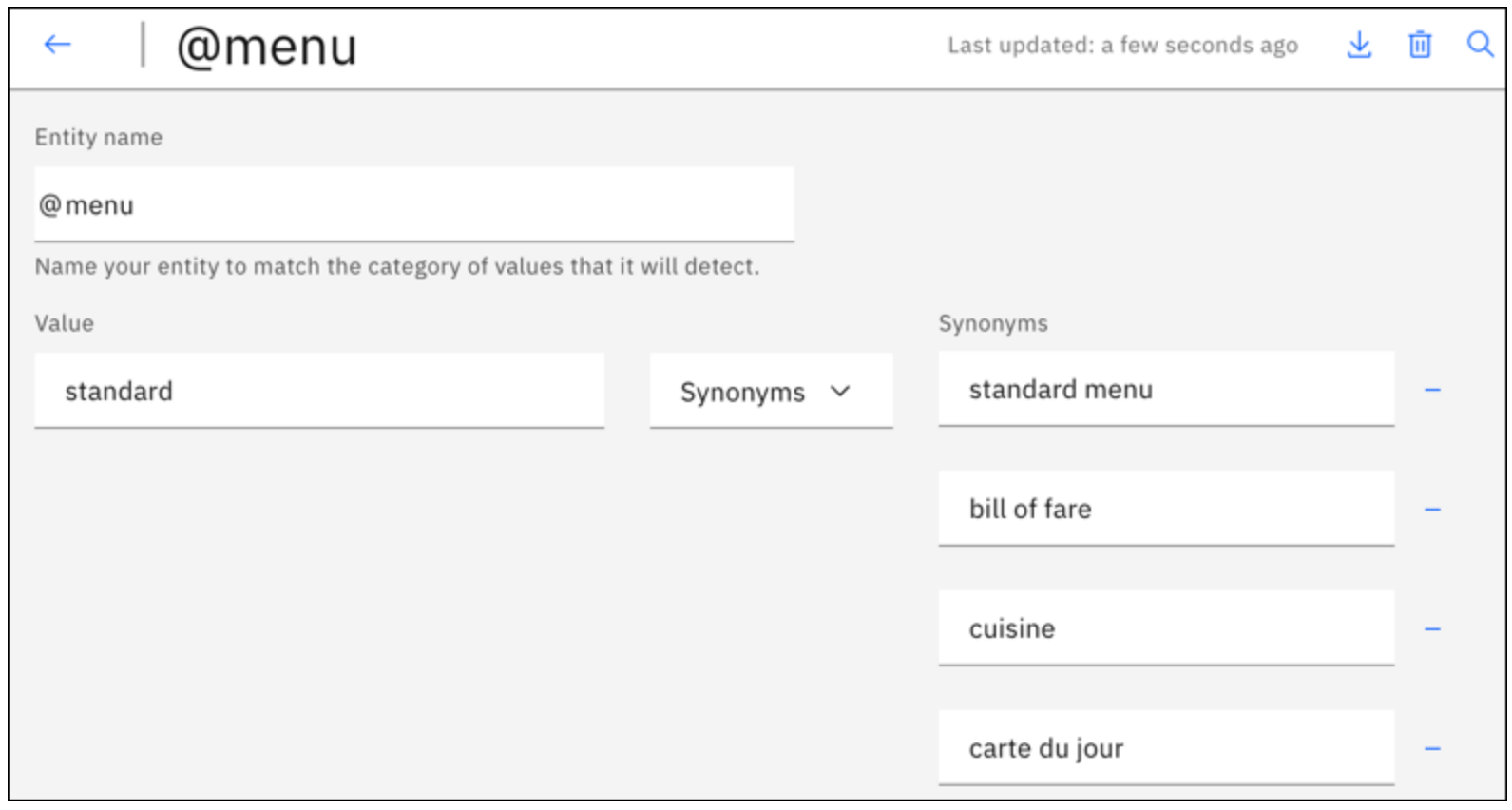
Options response

13. Click the **Close** icon to close the node edit view.

Add a @menu entity

To recognize the different types of menus that customers indicate they want to see, you add a `@menu` entity. Entities represent a class of object or a data type that is relevant to a user's purpose. By checking for the presence of specific entities in the user input, you can add more responses, each one tailored to address a distinct user request. In this case, you add a `@menu` entity that can distinguish among different menu types.

1. Click **My entities** to open the Entities page.
2. Click **Create entity**.
3. Enter `menu` into the entity name field.
4. Click **Create entity**.
5. Add `standard` to the **Value name** field, and then add `standard menu` to the **Synonyms** field, and press Enter.
6. Add the following additional synonyms:
   - `bill of fare`
   - `cuisine`
   - `carte du jour`



@menu entity

7. Click **Add value** to add the `@menu:standard` value.
8. Add `vegetarian` to the **Value name** field, and then add `vegetarian menu` to the **Synonyms** field, and press Enter.
9. Add the following additional synonyms:
   - `vegan diet`
   - `vegan`
   - `plants-only`
10. Click **Add value** to add the `@menu:vegetarian` value.
11. Add `cake` to the **Value name** field, and then add `cake menu` to the **Synonyms** field, and press Enter.
12. Add the following additional synonyms:
   - `cake shop menu`
   - `dessert menu`
   - `bakery offerings`
13. Click **Add value** to add the `@menu:cake` value.
14. Click the **Close** arrow ← to finish adding the `@menu` entity.

Add child nodes that are triggered by the @menu entity types

In this step, you add child nodes to the dialog node that checks for the `#menu` intent. Each child node shows a different response depending on the `@menu` entity type that the user chooses from the options list.

1. Click **Dialog** to open the dialog tree.
2. Find the **#menu** node in the dialog tree.

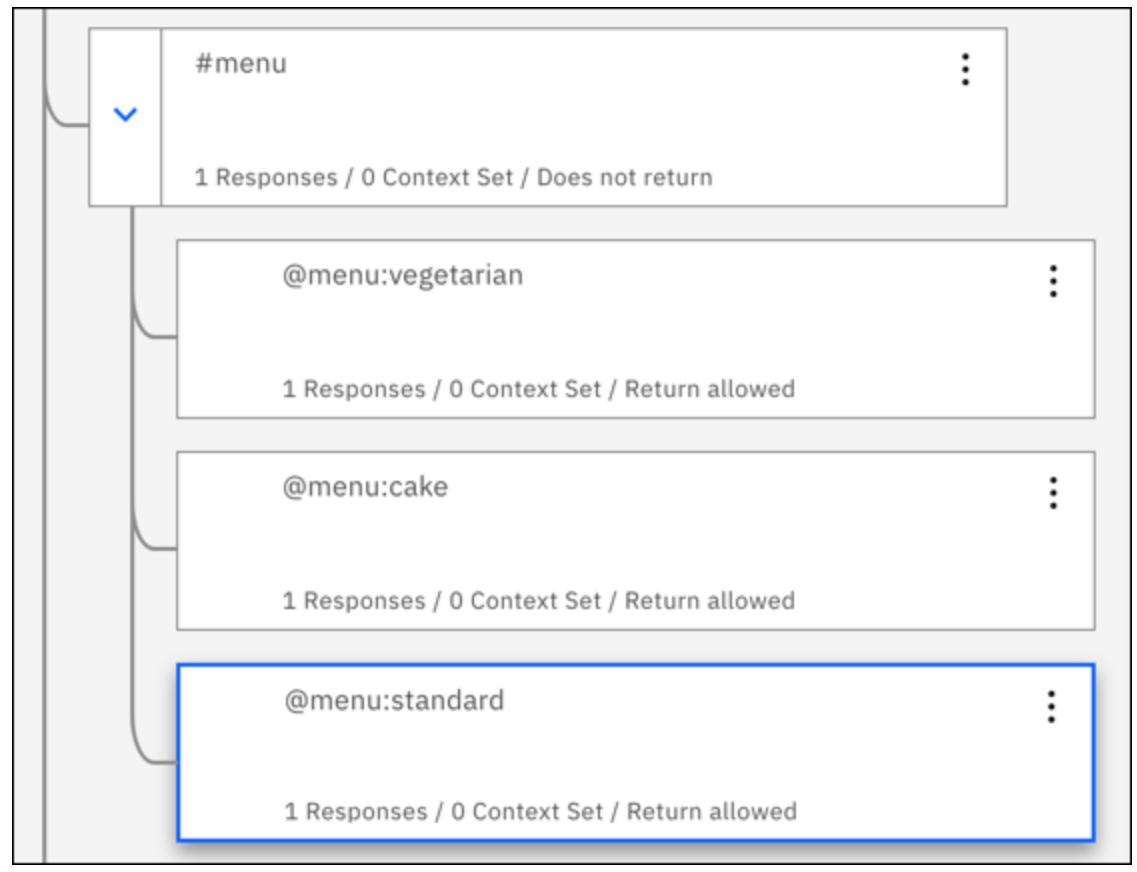
   You add a child node to handle each menu type option that you added to the **#menu** node.
3. Click the Node options icon ⋮ on the **#menu** node, and then select **Add child node**.
4. Start typing `@menu:standard` in the **If assistant recognizes** field, and then select it from the list.

   This node is used if the user input matches the `@menu:standard` entity.
5. In **Assistant responds**, enter the text response:

   ```
   To see our menu, go to the <a href="https://www.example.com/menu.html" target="blank">menu</a> page on our website.
   ```
6. Click the **Close** icon ⌄ to close the node edit view.
7. Click the Node options icon ⋮ on the **@menu:standard** node, and then select **Add node below**.
8. Start typing `@menu:vegetarian` into the **If assistant recognizes** field of this node, and then select it from the list.
9. In **Assistant responds**, enter the text response:

   ```
   To see our vegetarian menu, go to the <a href="https://www.example.com/vegetarian-menu.html" target="blank">vegetarian menu</a> page on our website.
   ```
10. Click the **Close** icon ⌄ to close the node edit view.
11. Click the Node options icon ⋮ on the **@menu:vegetarian** node, and then select **Add node below**.
12. Start typing `@menu:cake` into the **If assistant recognizes** field of this node, and then select it from the list.
13. In **Assistant responds**, enter the text response:

   ```
   To see our cake shop menu, go to the <a href="https://www.example.com/menu.html" target="blank">cake shop menu</a> page on our website.
   ```
14. Click the **Close** icon ⌄ to close the node edit view.



@menu nodes

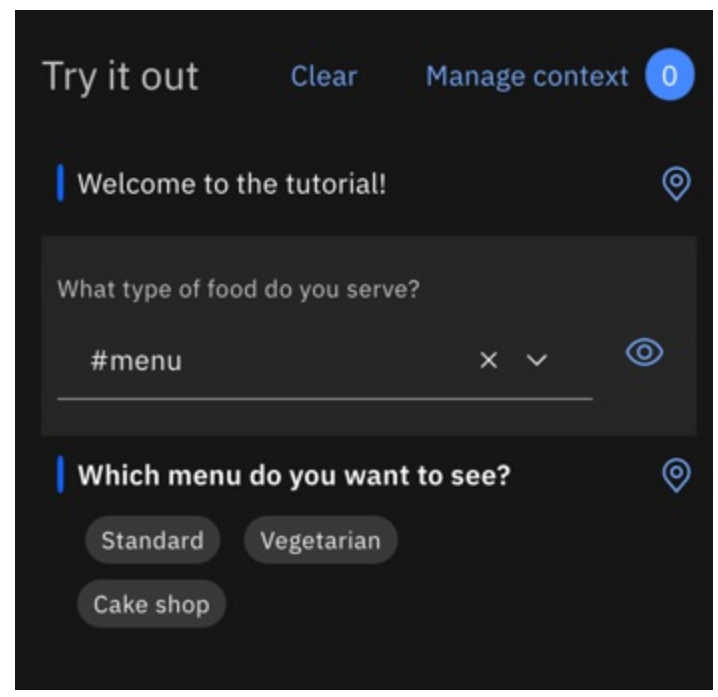
You added nodes that recognize user requests for menu details. Your response informs the user that three types of menus available, and asks them to choose one. When the user chooses a menu type, a response is displayed that provides a hypertext link to a web page with the requested menu details.

Test the menu options dialog nodes

Test the dialog nodes that you added to recognize menu questions.

1. Click **Try it**.
2. Type `What type of food do you serve?`

   Your assistant indicates that the `#menu` intent is recognized, and displays the list of menu options for the user to choose from.



Try it out

3. Click the `Cake shop` option.

   Your assistant recognizes the `#menu` intent and `@menu:cake` entity reference, and displays the response `To see our cake shop menu, go to the cake shop page on our website.`.

   A new web browser page opens and displays the example.com website.

4. Close the example.com web page.

You added an intent and entity that can recognize user requests for menu details, and can direct users to the appropriate menu.

The `#menu` intent represents a common, key need of potential restaurant customers. Due to its importance and popularity, you added a more complex section to the dialog to address it well.

Step 4: Manage cake orders

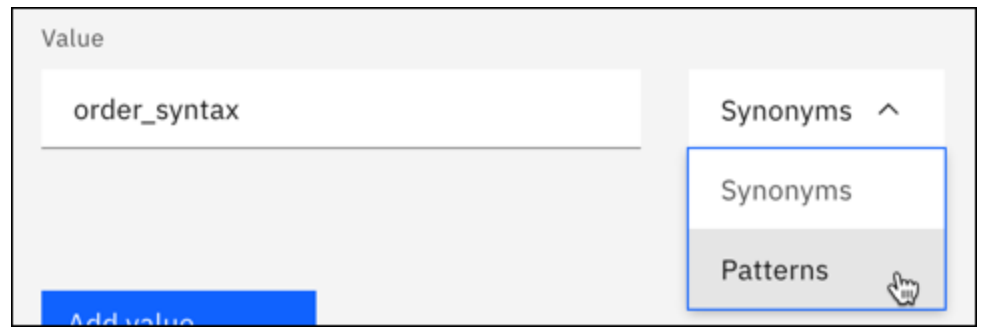
Customers can place orders in person, over the phone, or by using the order form on the website. After the order is placed, users can cancel the order through the virtual assistant. First, define an entity that can recognize order numbers. Then, add an intent that recognizes when users want to cancel a

cake order.

Adding an order number pattern entity

You want the assistant to recognize order numbers, so you create a pattern entity to recognize the unique format that the restaurant uses to identify its orders. The syntax of order numbers that are used by the restaurant's bakery is two uppercase letters followed by 5 numbers. For example, `YR34663`. Add an entity that can recognize this character pattern.

1. Click **My entities** to open the Entities page.
2. Click **Create entity**.
3. Enter `order_number` into the entity name field.
4. Click **Create entity**.
5. Add `order_syntax` to the **Value name** field.
6. Click the down arrow next to **Synonyms** to change the type to **Patterns**.



Pattern

7. Add the following regular expression to the Pattern field:

```
[A-Z]{2}\d{5}
```

8. Click **Add value**.
9. Click the **Close** arrow ← to finish adding the `@order_number` entity.

Add a cancel order intent

1. On the **Intents** page, click **Create intent**.
2. Add the following intent name, and then click **Create intent**:

```
cancel_order
```

3. In the **Add user examples** field, enter the following utterances:

```
I want to cancel my cake order
I need to cancel an order I just placed
Can I cancel my cake order?
I'd like to cancel my order
There's been a change. I need to cancel my bakery order.
please cancel the birthday cake order I placed last week
The party theme changed; we don't need a cake anymore
that order i placed, i need to cancel it.
```

4. Click the **Close** arrow ← to finish adding the `#cancel_order` entity.

Add a yes intent

Before you complete a task on the user's behalf, you must get confirmation that you are taking the proper action. Add a #yes intent to the dialog that can recognize when a user agrees with what your assistant is proposing.

1. On the **Intents** page, click **Create intent**.
2. Add the following intent name, and then click **Create intent**:

```
yes
```

3. In the **Add user examples** field, enter the following utterances:

```
Yes
Correct
Please do.
You've got it right.
Please do that.
that is correct.
That's right
yeah
Yup
Yes, I'd like to go ahead with that.
```

4. Click the **Close** arrow ← to finish adding the `#yes` intent.

Add dialog nodes that can manage requests to cancel an order

Now, add a dialog node that can handle requests to cancel a cake order.

1. Click **Dialog** to open the dialog tree.
2. Find the **#menu** node in the dialog tree.
3. Click the Node options icon ⋮ on the **#menu** node, and then select **Add node below**.
4. Start typing `#cancel_order` into the **If assistant recognizes** field of this node, then select it from the list.
5. In **Assistant responds**, enter the text response:

```
If the pickup time is more than 48 hours from now, you can cancel your order.
```

Before you can cancel the order, you need to know the order number. The user might specify the order number in the original request. So, to avoid asking for the order number again, check for a number with the order number pattern in the original input. To do so, define a context variable that would save the order number if it is specified.

1. In **Assistant responds**, click the **Options** icon ⋮ icon, and then select **Open context editor**.
2. Set the context by using these values:

Variable	Value
`$ordernumber`	`<? @order_number.literal ?>`
	Order number context variable details

The context variable value (`<? @order_number.literal ?>`) is a SpEL expression that captures the number that the user specifies that matches the pattern that is defined by the @order_number pattern entity. It saves it to the `$ordernumber` variable.

3. Click the **Close** icon ⌄ to close the node edit view.

Now, add child nodes that either ask for the order number or get confirmation from the user that they want to cancel an order with the detected order number.

1. Click the Node options icon ⋮ on the **#cancel_order** node, and then select **Add child node**.
2. Add a label to the node to distinguish it from other child nodes you are adding. In the **Enter node name** field, add `Ask for order number`.
3. In **If assistant recognizes**, enter `true` as the condition.
4. In **Assistant responds**, enter the text response:

```
What is the order number?
```

5. Click the **Close** icon ⌄ to close the node edit view.

Now, add another child node that informs the user that you are canceling the order.

1. Click the Node options icon ⋮ on the **Ask for order number** node, and then select **Add child node**.
2. In **If assistant recognizes**, enter `@order_number` as the condition.

3. In **Assistant responds**, click the **Options** icon ⋮ icon, and then select **Open context editor**.
4. Set the context by using these values:

Variable	Value
$ordernumber	<? @order_number.literal ?>
	Order number context variable details

5. In **Assistant responds**, enter the text response:

```
OK. The order $ordernumber is canceled. We hope we get the opportunity to bake a cake for you sometime soon.
```

6. Click the **Close** icon ⌄ to close the node edit view.

Add another node to capture the case where a user provides a number, but it is not a valid order number.

1. Click the Node options icon ⋮ on the **@order_number** node, and then select **Add node below**.
2. In **If assistant recognizes**, enter `true` as the condition.
3. In **Assistant responds**, enter the text response:

```
I need the order number to cancel the order for you. If you don't know the order number, please call us to cancel over the phone.
```

4. Click the **Close** icon ⌄ to close the node edit view.

Add a node after the initial order cancellation request node that responds in the case where the user provides the order number in the initial request, so you don't have to ask for it again.

1. Click the Node options icon ⋮ on the **#cancel_order** node, and then select **Add child node**.
2. Add a label to the node to distinguish it from other child nodes. In the **Enter node name** field, add `Number provided`.
3. In **If assistant recognizes**, enter `@order_number` as the condition.
4. In **Assistant responds**, enter the text response:

```
Just to confirm, you want to cancel order $ordernumber?
```

5. Click the **Close** icon ⌄ to close the node edit view.

You must add child nodes that check for the user's response to your confirmation question.

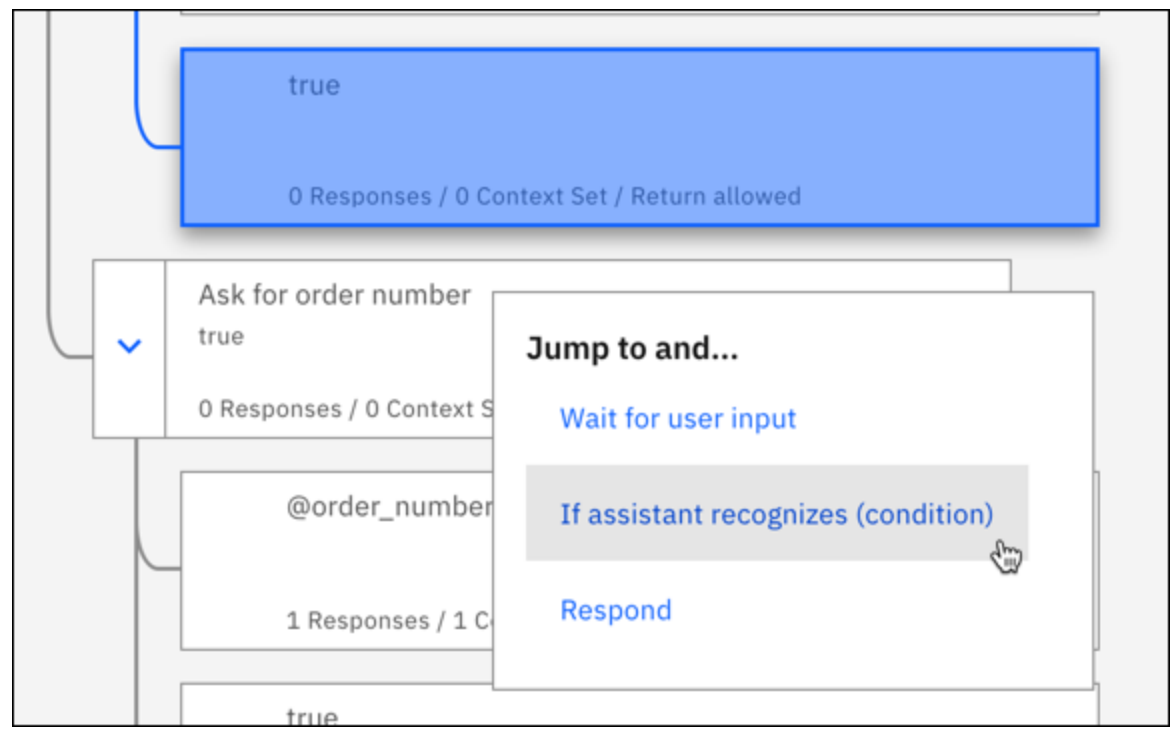
1. Click the Node options icon ⋮ on the **Number provided** node, and then select **Add child node**.
2. In **If assistant recognizes**, enter `#yes` as the condition.
3. In **Assistant responds**, enter the text response:

```
OK. The order $ordernumber is canceled. We hope we get the opportunity to bake a cake for you sometime soon.
```

4. Click the **Close** icon ⌄ to close the node edit view.
5. Click the Node options icon ⋮ on the **#yes** node, and then select **Add node below**.
6. In **If assistant recognizes**, enter `true` as the condition.

Don't add a response to **Assistant responds**. Instead, you redirect users to the branch that asks for the order number details that you created earlier.

1. In **Then assistant should** choose **Jump to**.
2. Select the **Ask for order number** node.
3. Choose **If assistant recognizes (condition)**.



Jump to

4. Click the **Close** icon to close the node edit view.

Force the conversation to evaluate the child nodes under the *#cancel_order* node at run time.

1. Click to open the `#cancel_order` node in the edit view.
2. In **Then assistant should** select `Skip user input`.
3. Click the **Close** icon to close the node edit view.

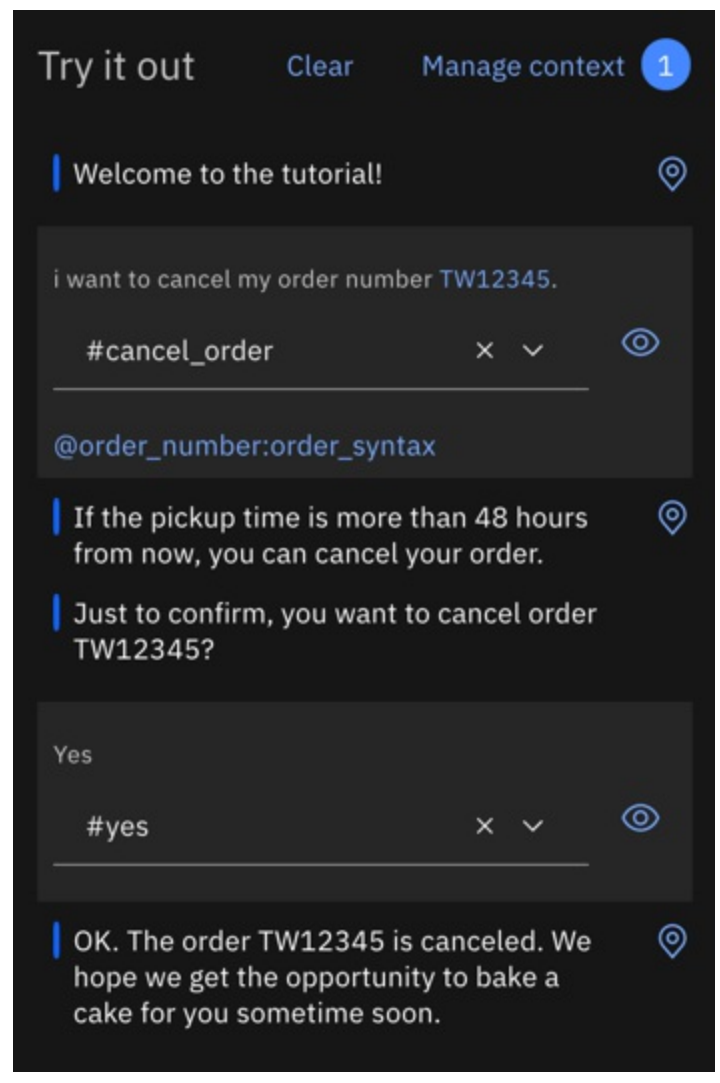
Test order cancellations

Test whether your assistant can recognize character patterns that match the pattern that is used for product order numbers in user input.

1. Click **Try it**.
2. Type `i want to cancel my order number TW12345.`

   Your assistant recognizes both the `#cancel_order` intent and the `@order_number` entity. It responds with, `If the pickup time is more than 48 hours from now, you can cancel your order. Just to confirm, you want to cancel order TW12345?`.
3. Type `Yes`.

   Your assistant recognizes the `#yes` intent and responds with, `OK. The order TW12345 is canceled. We hope we get the opportunity to bake a cake for you sometime soon.`.



Try it out

Now, try it when you don't know the order number.

4. Click **Clear** in the "Try it out" pane to start over.
5. Type `I want to cancel my order.`

   Your assistant recognizes the `#cancel_order` intent, and responds with, `If the pickup time is more than 48 hours from now, you can cancel your order. What is the order number?`.
6. Enter, `I don't know.`

   Your assistant responds with, `I need the order number to cancel the order for you. If you don't know the order number, please call us to cancel over the phone.`.

Add nodes to clarify the order number format

If you do more testing, you might find that the dialog isn't helpful in scenarios where the user does not remember the order number format. The user might include only the numbers or the letters too, but forget that they are meant to be uppercase. So, it would be a nice touch to give them a hint in such cases, correct? If you want to be kind, add another node to the dialog tree that checks for numbers in the user input.

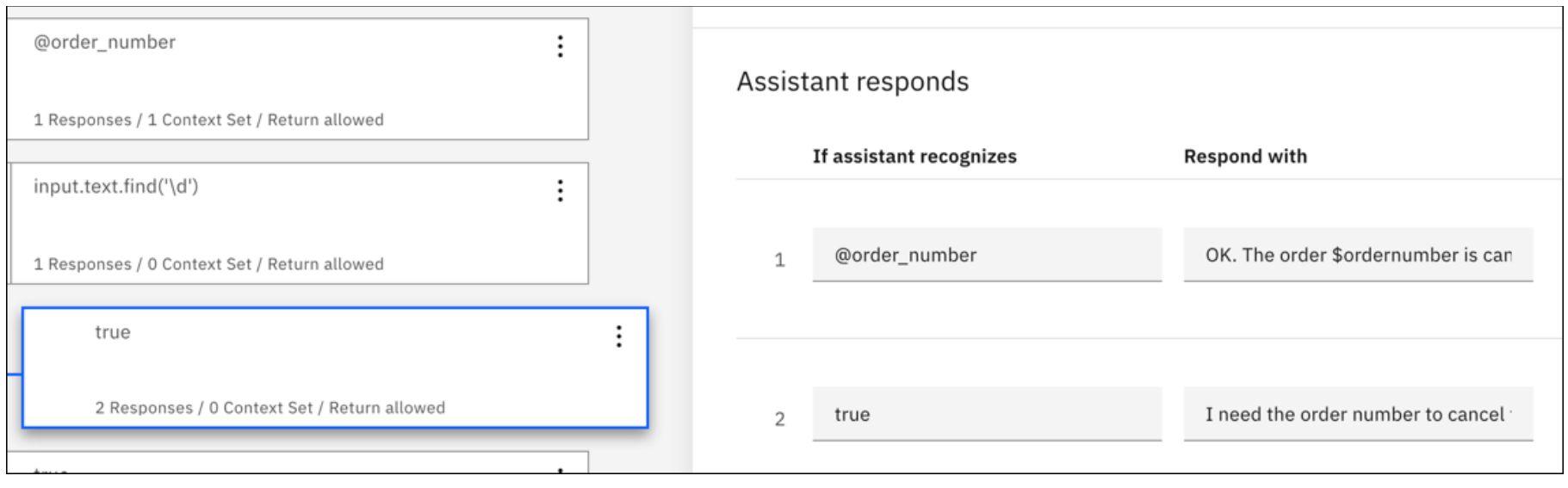
1. Click the Node options icon ⋮ on the **@order_number** node, and then select **Add node below**.
2. In **If assistant recognizes**, enter `input.text.find('\d')` as the condition.
3. In the condition field, add `input.text.find('\d')`, which is a SpEL expression that says if you find one or more numbers in the user input, trigger this response.
4. In **Assistant responds**, enter the text response:

   > The correct format for our order numbers is AAnnnnn. The A's represents 2 uppercase letters, and the n's represent 5 numbers. Do you have an order number in that format?

5. Click the **Close** icon ⌄ to close the node edit view.
6. Click the Node options icon ⋮ on the **input.text.find('\d')** node, and then select **Add child node**.
7. In **If assistant recognizes**, enter `true` as the condition.
8. Click **Customize**, set the **Multiple conditioned responses** switch to **On**, and then click **Apply**.
9. In **Assistant responds, add responses with the following conditions:

If assistant recognizes	Assistant responds	Then assistant should
@order_number	OK. The order $ordernumber is canceled. We hope we get the opportunity to bake a cake for you sometime soon.	Default to node settings
true	I need the order number to cancel the order for you. If you don't know the order number, please call us to cancel over the phone.	

Condition details



Condition details

10. Click the **Close** icon ⌄ to close the node edit view.

Now, when you test, you can provide a set of numbers or a mix of numbers and text as input, and the dialog reminds you of the correct order number

format. You tested your dialog, found a weakness in it, and corrected it.

> **Tip:** Another way that you can address this type of scenario is to add a node with slots. See the Adding a node with slots to a dialog tutorial to learn more about using slots.

Step 5: Add the personal touch

If the user shows interest in the bot itself, you want the virtual assistant to recognize that curiosity and engage with the user in a more personal way. You might remember the `#General_About_You` intent, which is provided with the *General* content catalog, that we considered using earlier before you added your own custom `#about_restaurant` intent. It is built to recognize such questions from the user. Add a node that conditions on this intent. In your response, you can ask for the user's name and save it to a $username variable that you can use elsewhere in the dialog, if available.

Add a node that handles questions about the bot

Add a dialog node that can recognize the user's interest in the bot, and respond.

1. Click **Dialog** to open the dialog tree.
2. Click the Node options icon ⋮ on the **Welcome** node, and then select **Add node below**.
3. Start typing `#General_About_You` in the **If assistant recognizes** field, and then select it from the list.
4. In **Assistant responds**, enter the text response:

   ```
   I am a virtual assistant that is designed to answer your questions about the Truck Stop Gourmand restaurant. What should I call you?
   ```

5. Click the **Close** icon ⌄ to close the node edit view.
6. Click the Node options icon ⋮ on the **#General_About_You** node, and then select **Add child node**.
7. In **If assistant recognizes**, enter `true` as the condition.
8. In **Assistant responds**, enter the text response:

   ```
   Hello, <? input.text ?>! It's lovely to meet you. How can I help you today?
   ```

To capture the name that the user provides, add a context variable to the node.

1. In **Assistant responds**, click the options menu ⋮ and select **Open context editor**.
2. Set the context by using these values:

Variable	Value
$username	"<? input.text ?>"

Set context

   The context variable value (`<? input.text ?>`) is a SpEL expression that captures the username as it is specified by the user, and then saves it to the `$username` context variable.

3. Click the **Close** icon ⌄ to close the node edit view.

If at run time, the user triggers this node and provides a name, then you record the user's name. If you know it, you should use it! Add conditional responses to the greeting dialog node that you added previously to include a conditional response that uses the username, if it is known.

Add the username to the greeting

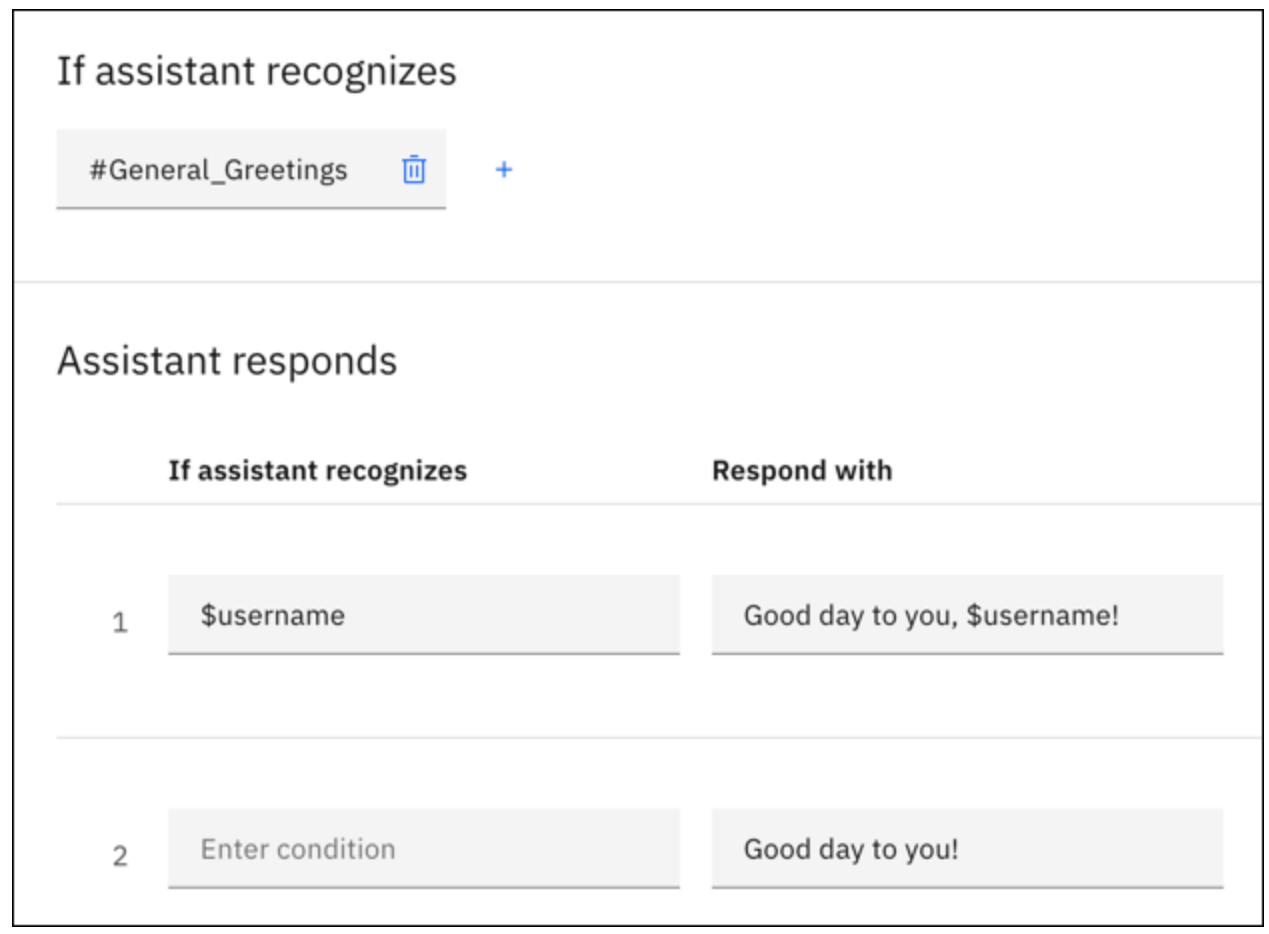
If you know the user's name, you should include it in your greeting message. To do so, add conditional responses, and include a variation of the greeting that includes the user's name.

1. Find the **Greet customers** node in the dialog tree, and click to open it in the edit view.
2. Click **Customize**, set the **Multiple conditioned responses** switch to **On**, and then click **Apply**.
3. Click **Add response**.
4. In **If assistant recognizes**, enter `$username`.

5. In **Respond with**, enter a text response:

   ```
   Good day to you, $username!
   ```

6. Click the **Move up** arrow to reorder the two response types.



Move up response

7. Click the **Close** icon to close the node edit view.

Test personalization

Test whether your assistant can recognize and save a user's name, and then refer to the user by it later.

1. Click **Try it**.
2. Click **Clear** to restart the conversation session.
3. Enter `Who are you?`

   Your assistant recognizes the `#General_About_You` intent. Its response ends with the question, `What should I call you?`.

4. Enter `Jane`.

   Your assistant saves `Jane` in the `$username` variable and responds `Hello, Jane! It's lovely to meet you. How can I help you today?`.

5. To test the `Greet customers` node, enter `Hello.`.

   Your assistant recognizes the `Greet customers` intent and says, `Good day to you, Jane!`. It uses the conditional response that includes the user's name because the `$username` context variable contains a value at the time that the greeting node is triggered.

You can add a conditional response that conditions on and includes the user's name for any other responses where personalization would add value to the conversation.

Summary

You created a more complex dialog that reacts to questions and requests from the customer.

Adding a node with slots to a dialog

In this tutorial, you add slots to a dialog node to collect multiple pieces of information from a user within a single node. The node that you create collects the information that is needed to make a restaurant reservation.

Learning objectives

By the time you finish the tutorial, you can learn how to:

- Define the intents and entities that are needed by your dialog

- Add slots to a dialog node
- Test the node with slots

Duration

This tutorial takes approximately 30 minutes to complete.

Prerequisite

Before you begin, complete the Getting Started tutorial. You use the dialog that you created, and add nodes to the simple dialog that you built as part of the getting started exercise.

Step 1: Add intents and examples

Add an intent, which is the purpose or goal that is expressed in user input. You add a #reservation intent that recognizes user input that indicates that the user wants to make a restaurant reservation.

1. On the **Intents** page, click **Create intent**.
2. Add the following intent name, and then click **Create intent**:

   ```
   reservation
   ```

   The #reservation intent is added. A number sign (`#`) prefix is added to the intent name to label it as an intent. This naming convention helps you and others recognize the intent as an intent. It has no example user utterances that are associated with it yet.
3. In the **Add user examples** field, type the following utterance, and then click **Add example**:

   ```
   i'd like to make a reservation
   ```
4. Add these additional examples to help your assistant recognize the `#reservation` intent.

   ```
   I want to reserve a table for dinner
   Can 3 of us get a table for lunch?
   do you have openings for next Wednesday at 7?
   Is there availability for 4 on Tuesday night?
   i'd like to come in for brunch tomorrow
   can i reserve a table?
   ```
5. Click the **Close** arrow ← to finish adding the `#reservation` intent and its example utterances.

Step 2: Add entities

An entity definition includes a set of entity *values* that represent vocabulary that is often used in the context of an intent. By defining entities, you can help your assistant identify references in the user input that are related to intents of interest. In this step, you enable system entities that can recognize references to time, date, and numbers.

1. Enable system entities that can recognize date, time, and number references in user input. Click **System entities**, and then turn on these entities:
   - `@sys-time`
   - `@sys-date`
   - `@sys-number`

You enabled the @sys-date, @sys-time, and @sys-number system entities. Now you can use them in your dialog.

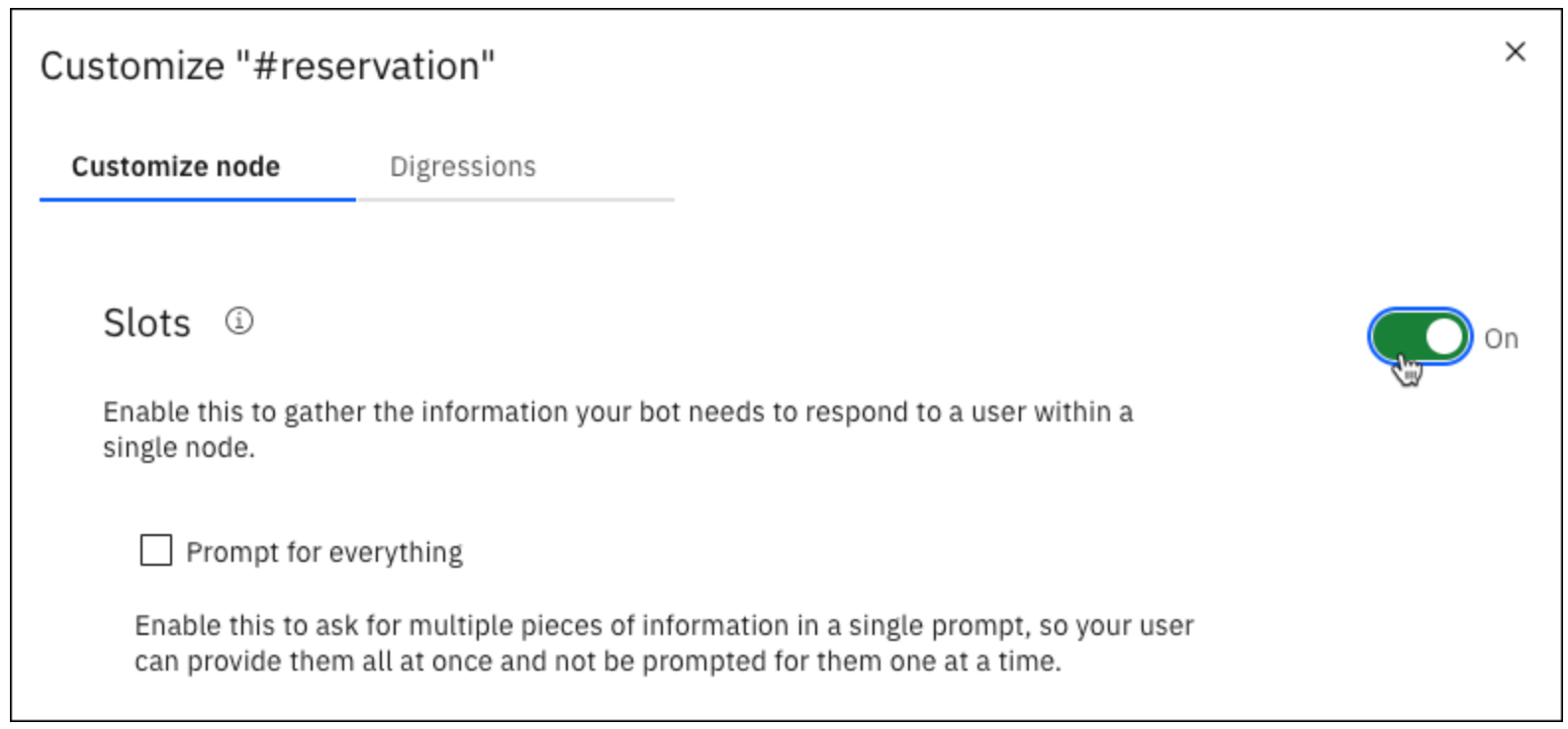
Step 3: Add a dialog node with slots

A dialog node represents the start of a thread of dialog between your assistant and the user. It contains a condition that must be met for the node to be processed by your assistant. At a minimum, it also contains a response. For example, a node condition might look for the `#hello` intent in user input, and respond with, `Hi. How can I help you?` This example is the simplest form of a dialog node, one that contains a single condition and a single response. You can define complex dialogs by adding conditional responses to a single node, adding child nodes that prolong the exchange with the user, and much more. (If you want to learn more about complex dialogs, you can complete the Building a complex dialog tutorial.)

The node that you add in this step is one that contains slots. Slots provide a structured format through which you can ask for and save multiple pieces of information from a user within a single node. They are most useful when you have a specific task in mind and need key pieces of information from the user before you can perform it. For more information, see Gathering information with slots.

The node that you add collects the information that is required to make a reservation at a restaurant.

1. Click **Dialog** to open the dialog tree.
2. Click the Node options icon ⋮ on the **Welcome** node, and then select **Add node below**.
3. Start typing `#reservation` in the **If assistant recognizes** field, and then select it from the list. This node is used if the user input matches the `#reservation` intent.
4. Click **Customize**, set the **Slots** switch to **On**, and then click **Apply**.



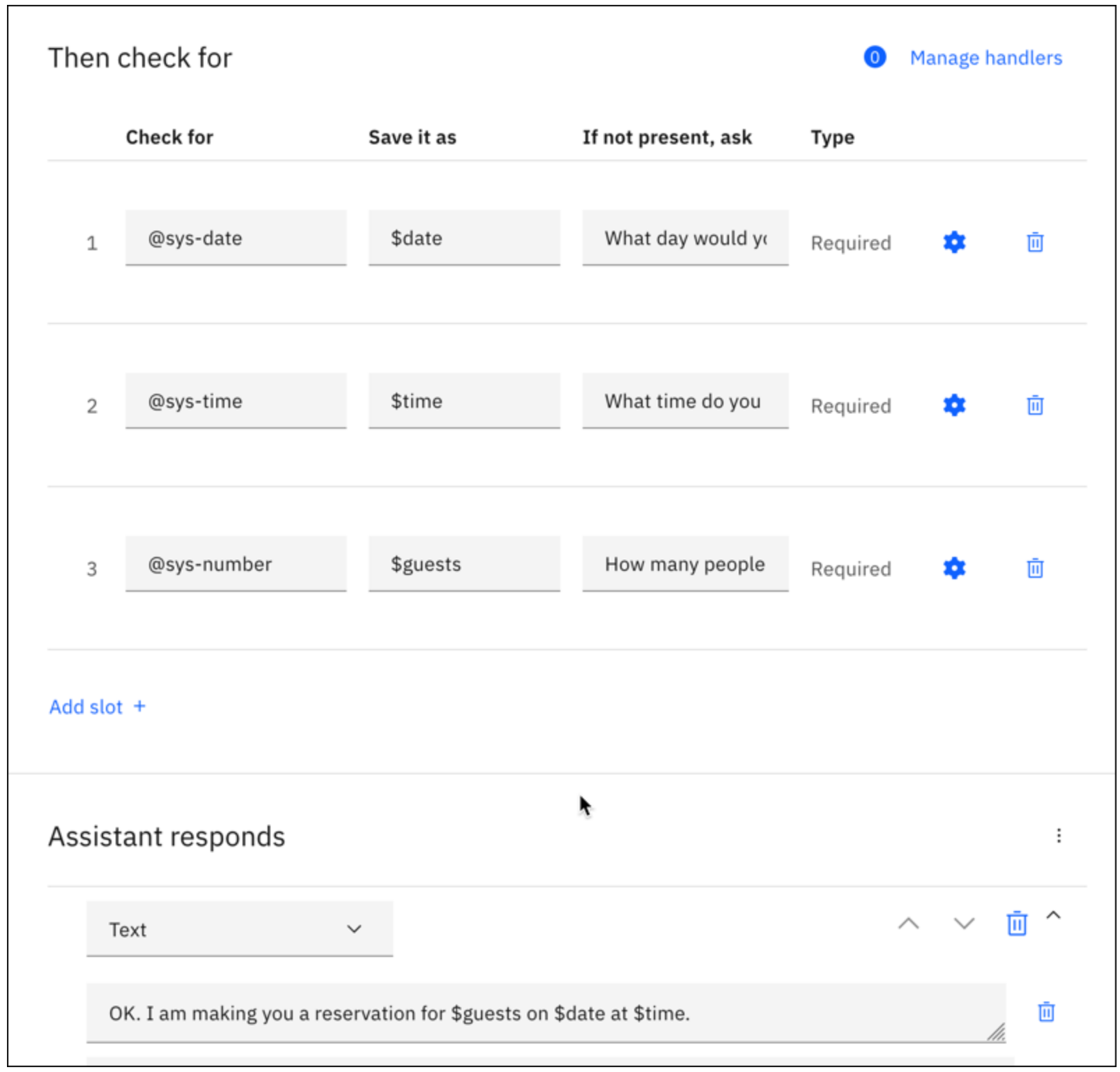
New dialog with built-in nodes

5. In the **Then check for** section, add the following slots:

Check for	Save it as	If not present, ask
@sys-date	$date	What day would you like to come in?
@sys-time	$time	What time do you want for the reservation?
@sys-number	$guests	How many people will be dining?

Slot details

6. In **Assistant responds**, enter the text response `OK. I am making you a reservation for $guests on $date at $time`.



Slots and response

7. Click the **Close** icon to close the node edit view.

Step 4: Test the dialog

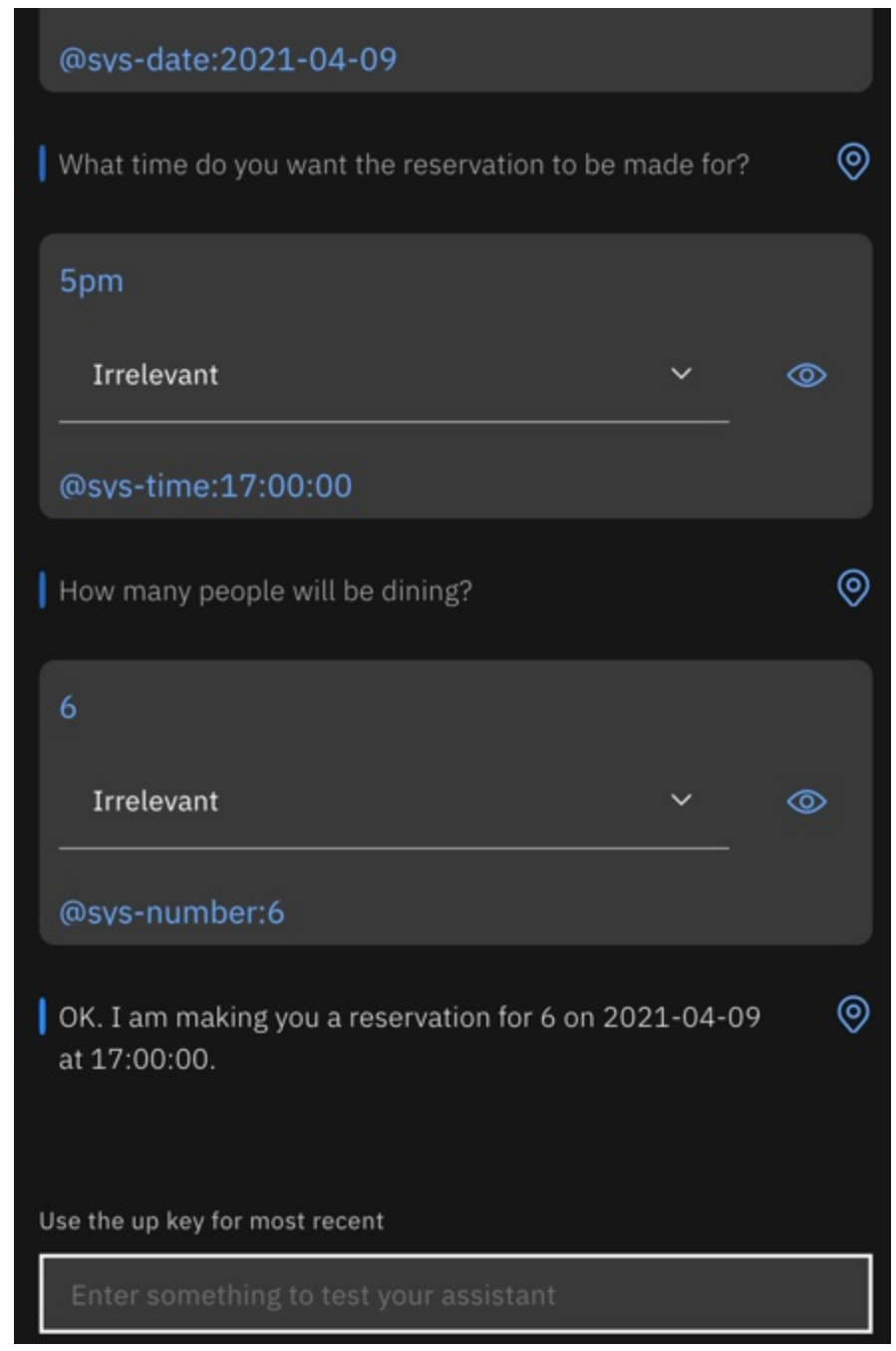
1. Click **Try it**.
2. Type `i want to make a reservation`.

   The assistant recognizes the #reservation intent, and it responds with the prompt for the first slot `What day would you like to come in?`.
3. Type `Friday`.

   The assistant recognizes the value, and uses it to fill the $date context variable for the first slot. It then shows the prompt for the next slot `What time do you want the reservation to be made for?`.
4. Type `5pm`.

   The assistant recognizes the value, and uses it to fill the $time context variable for the second slot. It then shows the prompt for the next slot, `How many people will be dining?`.
5. Type `6`.

   The assistant recognizes the value, and uses it to fill the $guests context variable for the third slot. Now that all of the slots are filled, it shows the node response, `OK. I am making you a reservation for 6 on 2021-04-09 at 17:00:00.`



Node slots

It worked! You created a node with slots.

Summary

You created a node with slots that can capture the information necessary to reserve a table at a restaurant.

Next steps

Improve the experience of users who interact with the node. Complete the follow-on tutorial, Improving a node with slots. It covers simple improvements, such as how to reformat the date (`2021-04-09`) and time (`17:00:00`) values that are returned by the system. It also covers more complex tasks, such as what to do if the user does not provide the type of value that your dialog expects for a slot.

Improving a dialog node with slots

In this tutorial, you enhance a simple node with slots that collects the information necessary to make a restaurant reservation.

Learning objectives

By the time you finish the tutorial, you learn how to:

- Test a node with slots
- Add slot response conditions that address common user interactions.
- Anticipate and address unrelated user input.
- Handle unexpected user responses.

Duration

This tutorial takes approximately 2 to 3 hours to complete.

Prerequisite

Before you begin, complete the Adding a node with slots to a dialog. You must complete the first slots tutorial before you begin this one because you build on the node with slots that you create in the first tutorial.

Step 1: Improve the format of the responses

When the date and time system entity values are saved, they are converted into a standardized format. This standardized format is useful for performing calculations on the values, but you might not want to show this reformatting to customers. In this step, you reformat the date (`2021-04-09`) and time (`17:00:00`) values that are referenced by the dialog.

To reformat the `$date` context variable value:

1. In your dialog, open the **#reservation** node.
2. Click the **Customize slot** icon for the `@sys-date` slot.
3. From the options menu ⋮ menu, select **Open JSON editor**, and then edit the JSON that defines the context variable. Add a method that reformats the date so that it converts the `2021-04-09` value into a full day of the week, followed by the full month and day. Edit the JSON as follows:

   ```
   {
     "context": {
       "date": "<? @sys-date.reformatDateTime('EEEE, MMMM d') ?>"
     }
   }
   ```

   The EEEE indicates that you want to spell out the day of the week. If you use 3 Es (EEE), the day of the week is shortened to Fri instead of Friday, for example. The MMMM indicates that you want to spell out the month. Again, if you use only 3 Ms (MMM), the month is shortened to Dec instead of December.
4. Click **Save**.
5. To change the format in which the time value is stored in the $time context variable to use the hour, minutes and indicate AM or PM, click the **Customize slot** icon for the `@sys-time` slot.
6. From the options menu ⋮ menu, select **Open JSON editor**, and then edit the JSON that defines the context variable so that it reads as follows:

   ```
   {
     "context": {
       "time": "<? @sys-time.reformatDateTime('h:mm a') ?>"
     }
   }
   ```
7. Click **Save**.
8. Test the node again. Open the "Try it out" pane, and click **Clear** to delete the slot context variable values that you specified when you tested the node with slots earlier. To see the impact of the changes you made, use the following script:

Speaker	Utterance
You	I want to make a reservation
Assistant	What day would you like to come in?
You	Friday
Assistant	What time do you want for the reservation?
You	5pm
Assistant	How many people will be dining?
You	6

Script details

This time the assistant responds with `OK. I am making you a reservation for 6 on Friday, April 9 at 5:00 PM.`

You improved the format that the dialog uses when it references context variable values in its responses. The dialog now uses `Friday, April 9` instead of the more technical, `2021-04-09`. And it uses `5:00 PM` instead of `17:00:00`.

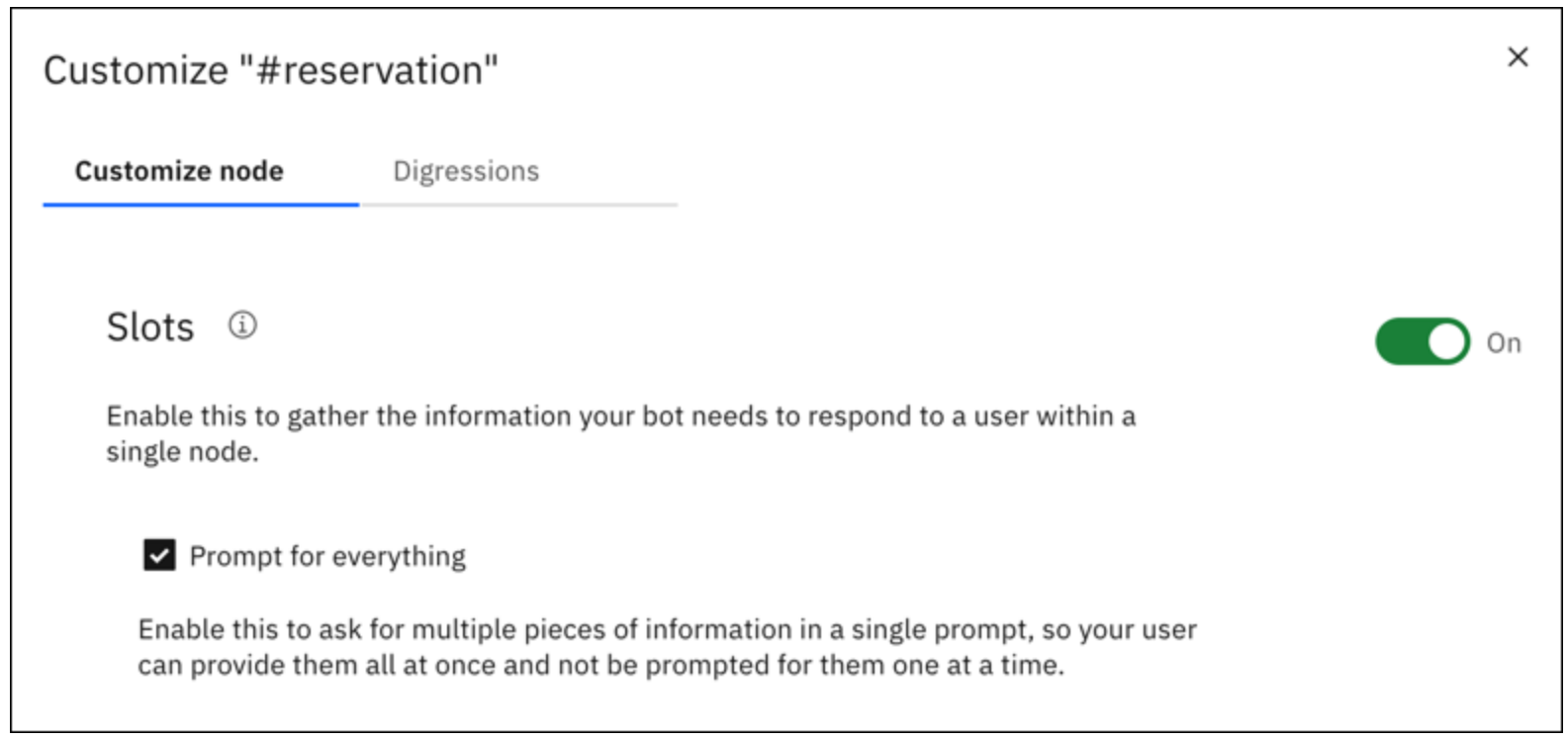
Step 2: Ask for everything at once

Now that you tested the dialog more than once, you might notice that it can be annoying to answer one slot prompt at a time. To prevent users from having to provide one piece of information at a time, you can ask for every piece of information that you need up front. Doing so gives the user a chance to provide all or some of the information in a single input.

The node with slots is designed to find and save any slot values that the user provides while the current node is being processed. You can help users to take advantage of the design by telling them what values to specify.

In this step, you learn how to prompt for everything at once.

1. Open the **#reservation** node.
2. Click **Customize**.
3. In **Slots**, select the **Prompt for everything** checkbox to enable the initial prompt, and then click **Apply**.

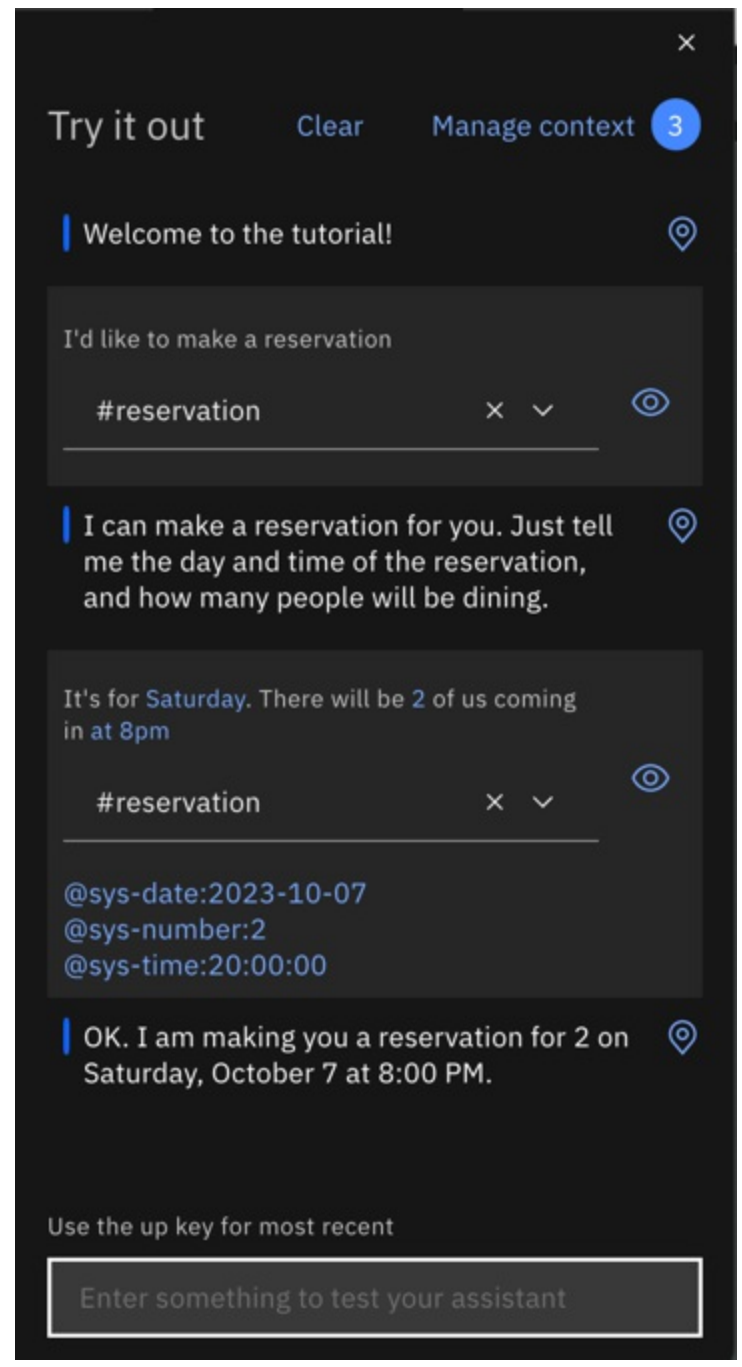


Prompt for everything

4. Back in the node edit view, scroll down to the newly added **If no slots are pre-filled, ask this first** section. Add the following initial prompt for the node, `I can make a reservation for you. Just tell me the day and time of the reservation, and how many people will be dining.`
5. Click the **Close** icon to close the node edit view.
6. Test this change from the "Try it out" pane. Open the pane, and then click **Clear** to nullify the slot context variable values from the previous test.
7. Enter `I'd like to make a reservation`.

   The dialog now responds with, `I can make a reservation for you. Just tell me the day and time of the reservation, and how many people it is for.`
8. Enter, `It's for Saturday. There will be 2 of us coming in at 8pm`.

   The assistant responds with, `OK. I am making you a reservation for 2 on Saturday at 8:00 PM.`



Test prompt for everything

Note: If the user provides any one of the slot values in their initial input, then the prompt that asks for everything is not displayed. For example, the initial input from the user might be `I want to make a reservation for this Friday night`. In this case, the initial prompt is skipped because you do not want to ask for information that the user already provided. The dialog shows the prompt for the next empty slot instead.

Step 3: Treat zeros properly

When you use the `sys-number` system entity in a slot condition, it does not deal with zeros properly. Instead of setting the context variable that you define for the slot to 0, your assistant sets the context variable to false. As a result, the slot does not think it is full and prompts the user for a number again and again until the user specifies a number other than zero.

1. Test the node so you can better understand the problem. Open the "Try it out" pane, and click **Clear** to delete the slot context variable values that you specified when you tested the node with slots earlier. Use the following script:

Speaker	Utterance
You	I want to make a reservation
Assistant	I can make a reservation for you. Just tell me the day and time of the reservation, and how many people will be dining.
You	We want to dine May 23 at 8pm. There will be 0 guests.
Assistant	How many people will be dining?
You	0
Assistant	How many people will be dining?

Script details

You are stuck in this loop until you specify a number other than 0.

To ensure that the `@sys-number` slot treats zeros properly:

1. Click the **Customize slot** icon 
2. In **Check for**, enter `@sys-number >= 0`.
3. Now, you change the context variable that is stored for the number.

   The slot condition is used both to check the user input for a number value, and to store that number in a context variable. Now that you edited the condition in the **Check for** field, you find mention of the number zero.

   However, you want to save only the number, and store it in the context variable. From the options menu ⋮ menu, select **Open JSON editor**, and then edit the JSON that defines the context variable.

   The value looks like this:

   ```
   {
     "context": {
       "guests": "@sys-number >= 0"
     }
   }
   ```

   Change it to look like this:

   ```
   {
     "context": {
       "guests": "@sys-number"
     }
   }
   ```

4. Click **Save**.

   You must edit the context variable value in the JSON editor. Do not edit the value in the slot's **Check for** field. The **Check for** field must remain set to `@sys-number >= 0`. { :important}

   When you edit the value in the JSON editor, you are effectively changing only what to save in the context variable. However, you do not want to change what to look for in the input. These two values will be different.

5. Test the node again. Open the "Try it out" pane, and click **Clear** to delete the slot context variable values that you specified when you tested the node with slots earlier. To see the impact of the changes you made, use the following script:

Speaker	Utterance
You	I want to make a reservation.
Assistant	I can make a reservation for you. Just tell me the day and time of the reservation, and how many people will be dining.
You	We want to dine May 23 at 8pm. There will be 0 guests.

   Script details

   This time the assistant responds with, `OK. I am making you a reservation for 0 on Thursday, May 23 at 8:00 PM.`

You formatted the number slot so that it treats zeros properly. But, you might not want the node to accept a zero as a valid number of guests. You will learn how to validate values that are specified by users in the next step.

Step 4: Validate user input

So far, we assume that the user provides the appropriate value types for the slots. You can account for times when users might provide an invalid value by adding conditional responses to slots. In this step, you use conditional slot responses to perform the following tasks:

- Check that the date requested is not in the past.
- Check whether a requested reservation time falls within the seating time window.
- Confirm the user's input.
- Check that the number of guests provided is larger than zero.
- Indicate that you are replacing one value with another.

To validate the date, complete the following steps:

1. Click the **Customize slot** icon for the `@sys-date` slot.
2. From the options menu ⋮ menu, select **Enable condition**.
3. In the **Found** section, click **Add a response**, then click the **Customize handler** icon .
4. Add the following condition and response to check whether the date that the user specifies falls before today:

If assistant recognizes	Assistant responds	Then assistant should
`@sys-date.before(now())`	`You cannot make a reservation for a day in the past.`	Clear slot and prompt again

Condition details

5. Add a second conditional response that is displayed if the user provides a valid date. This type of simple confirmation tells the user that the response was understood.

If assistant recognizes	Assistant responds	Then assistant should
`true`	`$date it is.`	Move on

Condition details

To validate the time:

1. Click the **Customize slot** icon for the `@sys-date` slot.
2. From the options menu ⋮ menu, select **Enable condition**.
3. In the **Found** section, click **Add a response**, then click the **Customize handler** icon .
4. Add the following conditions and responses to check whether the time that the user specifies falls within the allowed time window:

If assistant recognizes	Assistant responds	Then assistant should
`@sys-time.after('21:00:00')`	`Our last seating is at 9 PM.`	Clear slot and prompt again
`@sys-time.before('09:00:00')`	`Our first seating is at 9 AM.`	Clear slot and prompt again

Condition details

5. Add a third conditional response that is displayed if the user provides a valid time that falls within the window. This type of simple confirmation tells the user that the response was understood.

If assistant recognizes	Assistant responds	Then assistant should
`true`	`Ok, the reservation is for $time.`	Move on

Condition details

You can add conditions to validate the number of guests.

- Check that the number of guests that are specified is larger than zero.
- Anticipate and address the case when the user changes the number of guests.

If at any point while the node with slots is being processed, the user changes a slot value, the corresponding slot context variable value is updated. However, it can be useful to tell the user that the value is being replaced to give clear feedback to the user and to give the user a chance to rectify it.

1. Click the **Customize slot** icon for the `@sys-number` slot.
2. From the options menu ⋮ menu, select **Enable condition**.
3. In the **Found** section, click **Add a response**, then click the **Customize handler** icon .
4. Add the following conditions and responses to validate the number of guests:

If assistant recognizes	Assistant responds	Then assistant should
@sys-number == 0	Please specify a number that is larger than 0.	Clear slot and prompt again
(event.previous_value != null) && (event.previous_value != event.current_value)	Ok, updating the number of guests from <? event.previous_value ?> to <? event.current_value ?>.	Move on
true	Ok. The reservation is for $guests guests.	Move on

Condition details

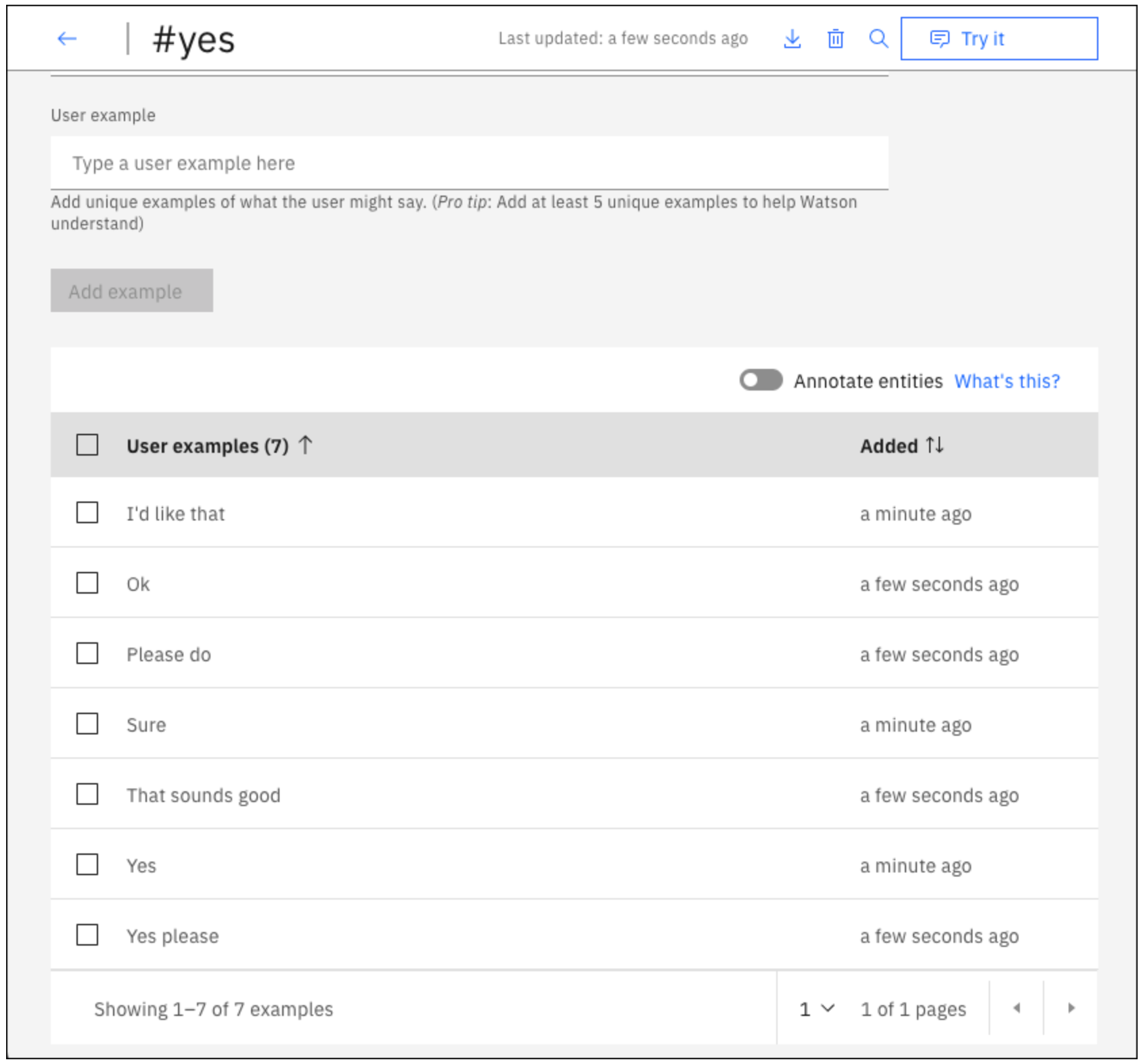
Step 5: Add a confirmation slot

You might want to design your dialog to call an external reservation system and book a reservation for the user in the system. Before your application completes this task, you probably want to confirm with the user that the dialog understands the details of the reservation correctly. You can do so by adding a confirmation slot to the node.

1. The confirmation slot expects a Yes or No answer from the user. You must teach the dialog to be able to recognize a Yes or No intent in the user input first.
2. Click the **Intents** tab to return to the Intents page.
3. Add the following intents and example utterances.

#yes

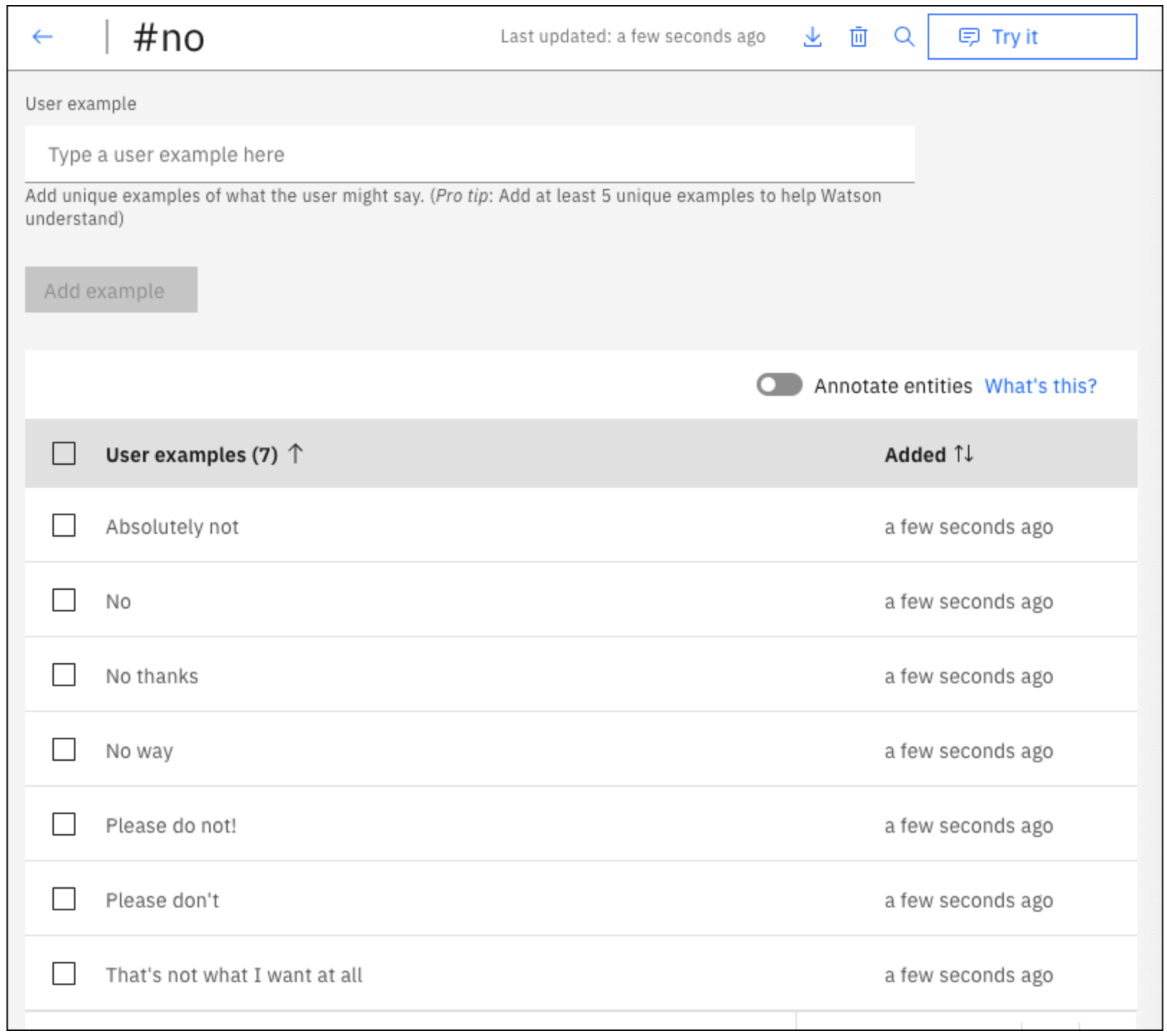
```
Yes
Sure
I'd like that
Please do
Yes please
Ok
That sounds good
```



Yes intent

#no

No
No thanks.
Please don't
Please do not!
That's not what I want at all
Absolutely not
No way



No intent

1. In your dialog, open the **#reservation** node.
2. In the **Then check for** section, click **Add slot** to add a fourth slot:

Check for	Save it as	If not present, ask
(#yes || #no) && slot_in_focus	$confirmation	I'm going to reserve you a table for $guests on $date at $time. Should I go ahead?

Slot details

This condition checks for either answer. You specify what happens next depending on whether the user answer Yes or No by using conditional slot responses. The `slot_in_focus` property forces the scope of this condition to apply to the current slot only. This setting prevents random statements that might match against a `#yes` or `#no` intent that the user might make from triggering this slot.

For example, the user might be answering the number of guests slot, and say something like, `Yes, there will be 5 of us.`. You do not want the `Yes` included in this response to accidentally fill the confirmation slot. By adding the `slot_in_focus` property to the condition, a `yes` or `no` answer from the user is applied to this slot only when the user is answering the prompt for this slot specifically.

3. Click the **Customize slot** icon for the `yes/no` slot.
4. From the options menu ⋮ menu, select **Enable condition**.
5. In the **Found** section, click **Add a response**, then click the **Customize handler** icon
6. Add the following condition and response to check for a `No` response:

If assistant recognizes	Assistant responds	Then assistant should
#no	Alright. Let's start over. I'll try to keep up this time.	Clear slot and prompt again

Condition details

7. When the `#no` intent is found you need to reset the context variables that you saved earlier, so you can ask for the information again. From the

options menu for the condition ⋮, select **Open context editor**.

8. Set the context by using these values:

Variable	Value
$date	null
$time	null
$guests	null

Set context

9. Click **Save**.
10. In the **Not found** section, clarify that you are expecting the user to provide a Yes or No answer.
11. Add a response with the following condition:

If assistant recognizes	Assistant responds	Then assistant should
true	Respond with Yes to indicate that you want the reservation to be made as-is, or No to indicate that you do not.	Wait for user input

Condition details

12. Click **Save**.

Now that you have confirmation responses for slot values, and you ask for everything at once, you might notice that the individual slot responses are displayed before the confirmation slot response is displayed, which can appear repetitive. Edit the slot found responses to prevent them from being displayed under certain conditions.

1. Click the **Customize slot** icon for the `@sys-date` slot.
2. In the **Found** section, replace the `true` condition. For example:

If assistant recognizes	Assistant responds	Then assistant should
!($time && $guests)	$date it is	Move on

Condition details

3. Click the **Customize slot** icon for the `@sys-time` slot.
4. In the **Found** section, replace the `true` condition. For example:

If assistant recognizes	Assistant responds	Then assistant should
!($date && $guests)	Ok, the reservation is for $time.	Move on

Condition details

5. Click the **Customize slot** icon for the `@sys-number` slot.
6. In the **Found** section, replace the `true` condition. For example:

If assistant recognizes	Assistant responds	Then assistant should
!($date && $time)	Ok. The reservation is for $guests guests.	Move on

Condition details

If you add more slots later, you must edit these conditions to account for the associated context variables for the additional slots. If you do not include a confirmation slot, you can specify `!all_slots_filled` only, and it would remain valid no matter how many slots you add later.

Step 6: Reset the slot context variable values

You might notice that before each test, you must clear the context variable values that were created during the previous test. You must do so because the node with slots prompts users only for information that it considers to be missing. If the slot context variables are all filled with valid values, no prompts are displayed. The same is true for the dialog at run time. You must build into the dialog a mechanism by which you reset the slot context variables to null so that the slots can be filled anew by the next user. To do so, you are going to add a parent node to the node with slots that sets the context variables to null.

1. From the tree view of the dialog, click the **Node options** icon ⋮ icon on the **#reservation** node with slots, and then select **Add node above**.
2. In **If assistant recognizes**, enter `#reservation` as the condition. The same condition that is used by the node with slots, but you change the condition for the node with slots later in this procedure.
3. In **Assistant responds**, click the options menu ⋮ and select **Open context editor**.
4. Set the context by using these values:

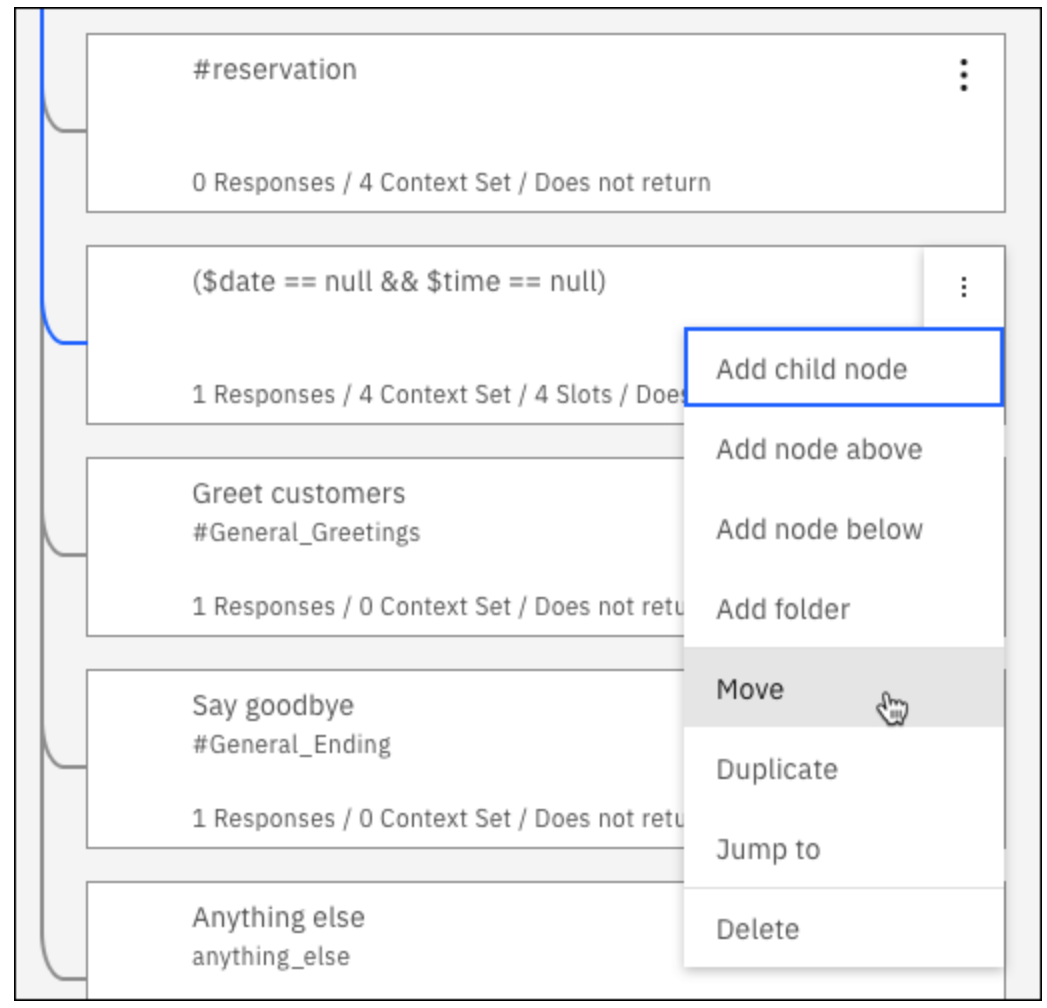
Variable	Value
$date	null
$time	null
$guests	null
$confirmation	null

Set context

5. Click to edit the other **#reservation** node, the one you created previously and to which you added the slots.
6. Change the node condition from `#reservation` to `($date == null && $time == null)`.
7. Click the **Node options** icon ⋮ icon on the node with slots, and then select **Move**.

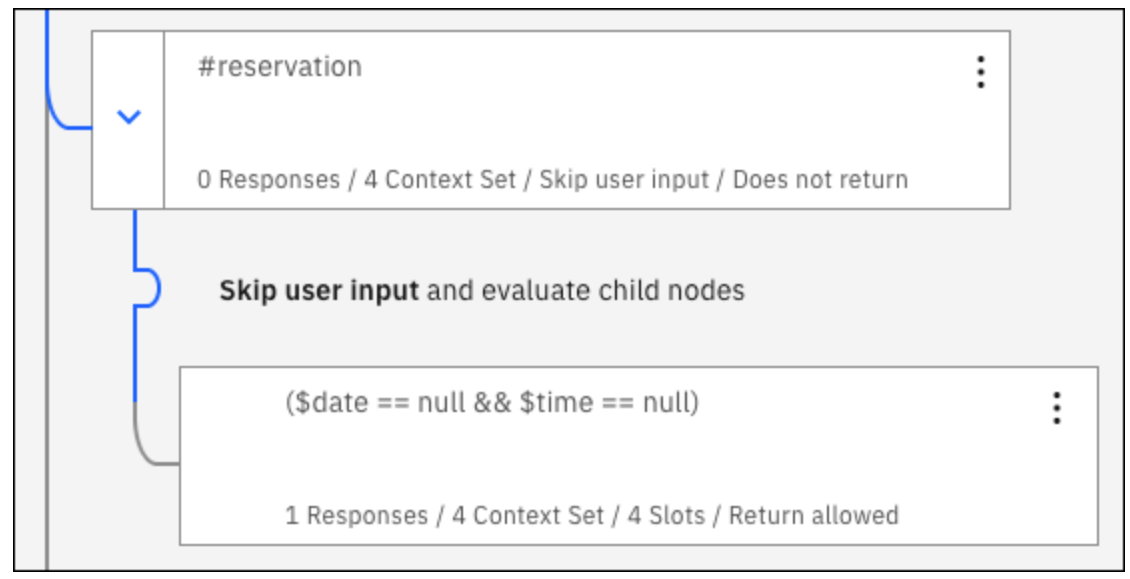
Move node

8. Select the `#reservation` node as the move-to location target, and then choose **As child node** from the menu.



Move as child node

9. Click to edit the **#reservation** node. In the **Then assistant should** section, change from **Wait for reply** to **Skip user input**.



Skip user input

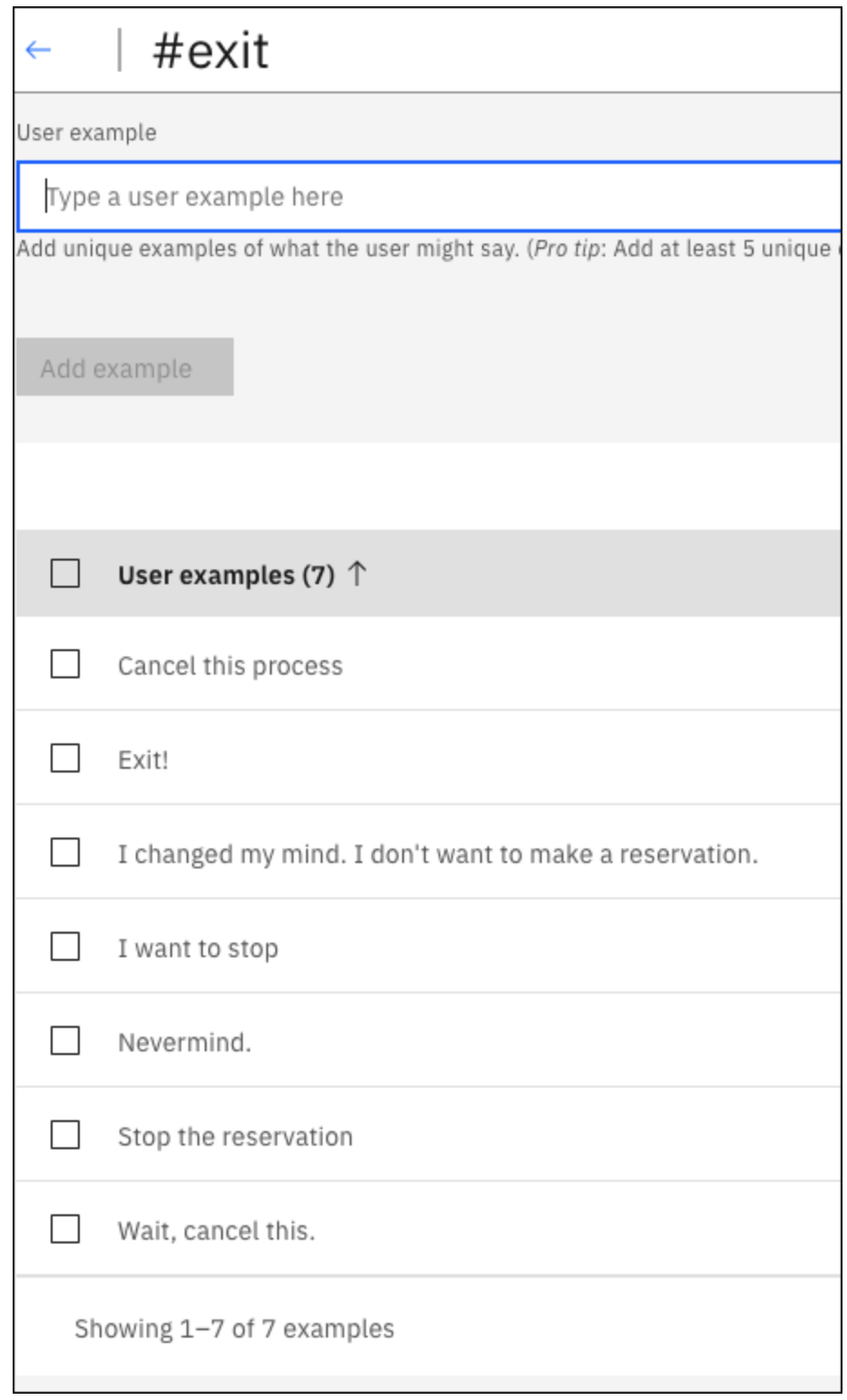
When a user input matches the `#reservation` intent, this node is triggered. The slot context variables are all set to null, and then the dialog jumps directly to the node with slots to process it.

Step 7: Give users a way to exit the process

Adding a node with slots is powerful because it keeps users on track providing the information that you need to give them a meaningful response or perform an action on their behalf. However, the user might be in the middle of providing reservation details, but decides to not go through with placing the reservation. You must give users a way to exit the process gracefully. You can do so by adding a slot handler that can detect a user's desire to exit the process, and exit the node without saving any values that were collected.

1. In **Intents**, add an `#exit` intent with the following example utterances.

```
I want to stop
Exit!
Cancel this process
I changed my mind. I don't want to make a reservation.
Stop the reservation
Wait, cancel this.
Never mind.
```



#exit intent

2. In **Dialog**, click to open the node with slots, and then click **Manage handlers**.
3. Add the following handler condition:

If assistant recognizes	Assistant responds	Then assistant should
#exit	Ok, we'll stop there. No reservation will be made.	Skip to response

Condition details

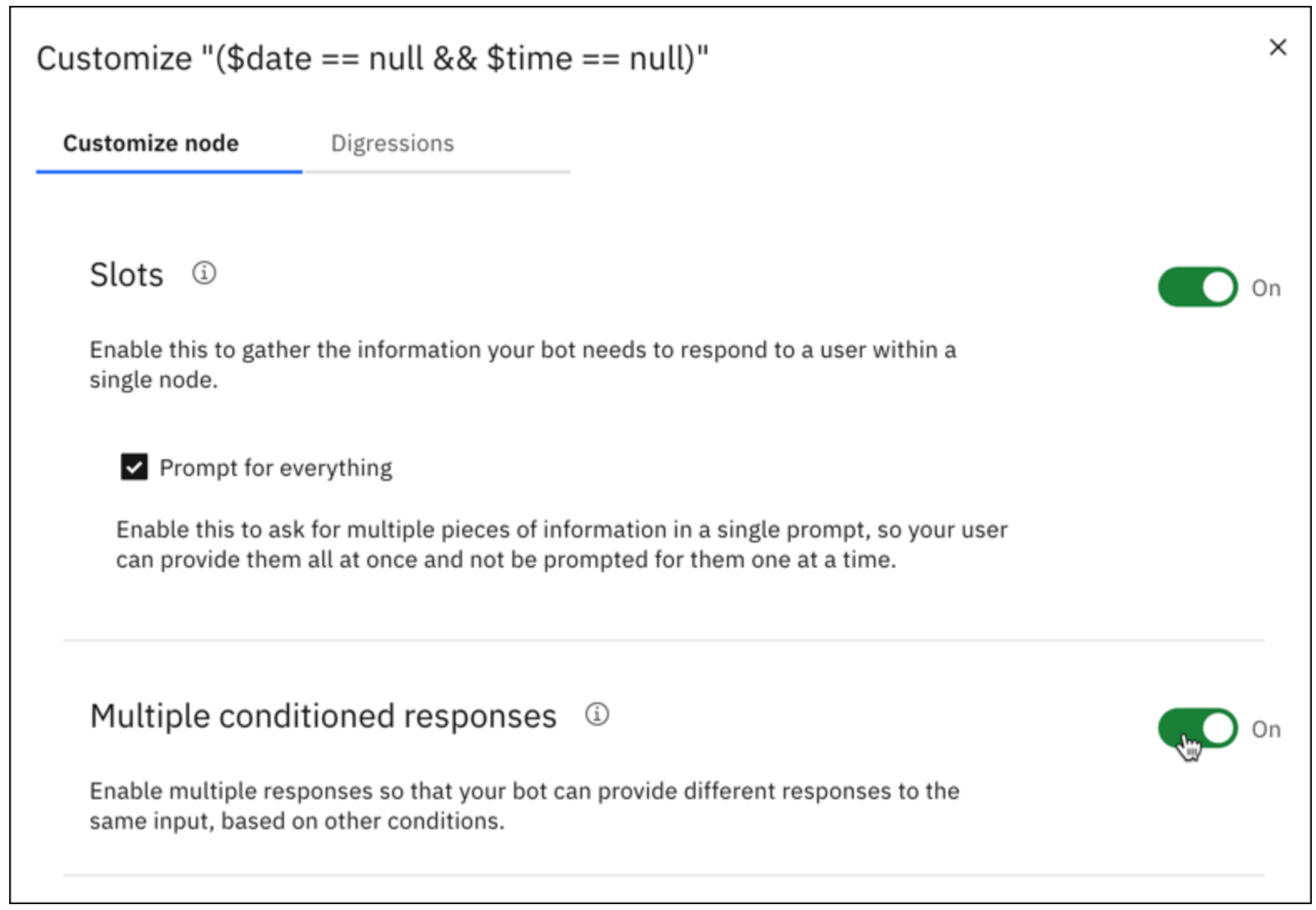
   The **Skip to response** action jumps directly to the node-level response without displaying the prompts associated with any of the remaining unfilled slots.

4. Click **Save**, and then click **Save** again.

Now, you need to edit the node-level response to have it recognize when a user wants to exit the process rather than make a reservation.

To add a conditional response for the node:

1. In the node with the slots, click **Customize**, set the **Multiple conditioned responses** switch to **On**, and then click **Apply**.



Multiple conditioned responses

2. In **Assistant responds**, click **Add response**.
3. Add a response with the following condition:

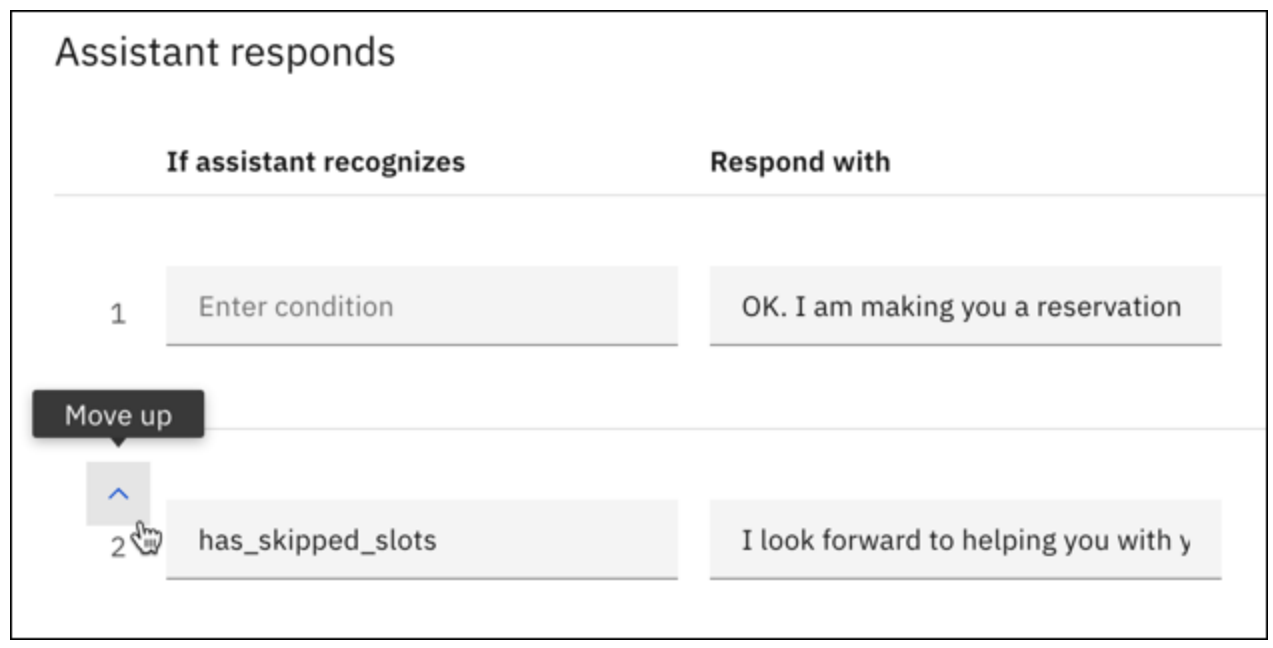
If assistant recognizes	Assistant responds	Then assistant should
has_skipped_slots	I look forward to helping you with your next reservation. Have a good day.	Default to node settings

Condition details

The `has_skipped_slots` condition checks the properties of the slots node to see if any of the slots were skipped. The `#exit` handler skips all remaining slots to go directly to the node response. So, when the `has_skipped_slots` property is present, you know the `#exit` intent was triggered, and the dialog can display an alternative response.

> **Note:** If you configure more than one slot to skip other slots, or configure another node-level event handler to skip slots, then you need a different approach to check if the #exit intent was triggered. See Handling requests to exit a process for an alternative way to do so.

4. You want your assistant to check for the `has_skipped_slots` property before it displays the standard node-level response. Move the `has_skipped_slots` conditional response so it gets processed before the original conditional response or it is never triggered. To do so, hover on the response you just added, use the **up arrow** to move it up, and then click **Save**.



Move conditional response

5. Test this change by using the following script in the "Try it out" pane.

Speaker	Utterance
You	I want to make a reservation.
Assistant	I can make a reservation for you. Just tell me the day and time of the reservation, and how many people will be dining.
You	It's for 5 people
Assistant	Ok. The reservation is for 5 guests. What day would you like to come in?
You	Never mind
Assistant	Ok, we'll stop there. No reservation will be made. I look forward to helping you with your next reservation. Have a good day.

Script details

Step 8: Apply a valid value if the user fails to provide one after several attempts

In some cases, a user might not understand what you are asking for. They might respond again and again with the wrong types of values. To plan for this possibility, you can add a counter to the slot, and after 3 failed attempts by the user to provide a valid value, you can apply a value to the slot on the user's behalf and move on.

For the `$time` information, you define a follow-up statement that is displayed when the user does not provide a valid time. Create a context variable that can track how many times the user provides a value that does not match the value type that the slot expects. You want the context variable to be initialized and set to `0` before the node with slots is processed, so you add it to the parent `#reservation` node.

1. In **Dialog**, click to edit the `#reservation` node.
2. In **Then set context**, click **Add variable**. (If **Then set context** isn't visible, click the options menu ⋮ in **Assistant responds** and select **Open context editor**.)
3. Set the context by using these values:

Variable	Value
`$counter`	`0`

Set context

4. Click the **Close** icon ⌄ to close the node edit view.
5. From the tree view, click to edit the node with slots.
6. Click the **Customize slot** icon ⚙ for the `@sys-time` slot.
7. Click the options menu ⋮ and select **Enable condition**. An **Enable this slot if:** section appears, set to `true`.
8. In the **Not found** section, add a response with the following condition:

If assistant recognizes	Assistant responds	Then assistant should
`true`	`Please specify the time that you want to eat. The restaurant seats people between 9AM and 9PM.`	Wait for user input

Condition details

Now add a 1 to the `counter` variable each time this response is triggered. This *Not found* response is triggered only when the user does not provide a valid time value.

1. In **Assistant responds**, click the options menu ⋮ and select **Open context editor**.
2. Set the context by using these values:

Variable	Value
`$counter`	`"<? context['counter'] + 1 ?>"`

Set context

3. Click **Save**.

Add a second conditional response to the **Not found** section that checks whether the counter is greater than 1, which indicates that the user provided an invalid response 3 times previously. In this case, the dialog assigns the time value on the user's behalf to the popular dinner reservation time of 8 PM. The user has a chance to change the time value when the confirmation slot is triggered.

1. In the **Not found** section, add a response with the following condition:

If assistant recognizes	Assistant responds	Then assistant should
`$counter > 1`	`You seem to be having trouble choosing a time. I will make the reservation at 8PM for you.`	Wait for user input

Condition details

2. You must set the $time variable to 8 PM. In **Assistant responds**, click the options menu ⋮ and select **Open context editor**.
3. Set the context by using these values:

Variable	Value
`$time`	`"<? '20:00:00'.reformatDateTime('h:mm a') ?>"`

Set context

4. Click **Save**.

The conditional response that you just added has a more precise condition than the true condition that is used by the first conditional response. You must move this response so it comes before the original conditional response or it is never triggered.

1. Hover on the response you just added, use the **up arrow** to move it up, and then click **Save**.
2. Test your changes by using the following script.

Speaker	Utterance
You	I want to make a reservation.
Assistant	I can make a reservation for you. Just tell me the day and time of the reservation, and how many people will be dining.
You	Tomorrow
Assistant	Friday, December 29 it is. What time do you want the reservation to be made for?
You	orange
Assistant	Please specify the time that you want to eat. The restaurant seats people between 9AM and 9PM.
You	pink
Assistant	Please specify the time that you want to eat. The restaurant seats people between 9AM and 9PM.
You	purple
Assistant	You seem to be having trouble choosing a time. I will make the reservation at 8 PM for you. How many people will be dining?

Script details

Step 9: Connect to an external service

Now that your dialog can collect and confirm a user's reservation details, you can call an external service to reserve a table in the restaurant's system or through a multi-restaurant online reservations service. See Making programmatic calls from a dialog node for more details.

In the logic that calls the reservation service, be sure to check for `has_skipped_slots` and do not continue with the reservation if it is present.

Summary

In this tutorial, you tested a node with slots and made changes that optimize how it interacts with real users. For more information about this subject, see Gathering information with slots.

Understanding digressions in dialog

In this tutorial, you see firsthand how digressions work.

Learning objectives

By the time you finish the tutorial, you learn how:

- Digressions are designed to work.
- Digression settings impact the flow of the dialog.
- To test digression settings for a dialog.

Duration

This tutorial takes approximately 20 minutes to complete.

Step 1: Step 1: Get the Digressions Showcase dialog

You use the *Digression Showcase* dialog as an example in this tutorial.

1. Download digression-showcase.json from the IBM GitHub repository.
2. Next, upload the JSON file to your assistant.

If you are using dialog in watsonx Assistant:

1. The upload overwrites any existing dialog. Use an assistant without an existing dialog, then activate dialog.
2. In **Dialog**, click **Upload/Download**.
3. Upload `digression-showcase.json`.

If you are using a dialog skill in the classic experience:

1. On **Skills**, click **Create skill**.
2. Choose **Dialog skill**, then click **Next**.
3. Choose **Upload skill**, then upload `digression-showcase.json`.

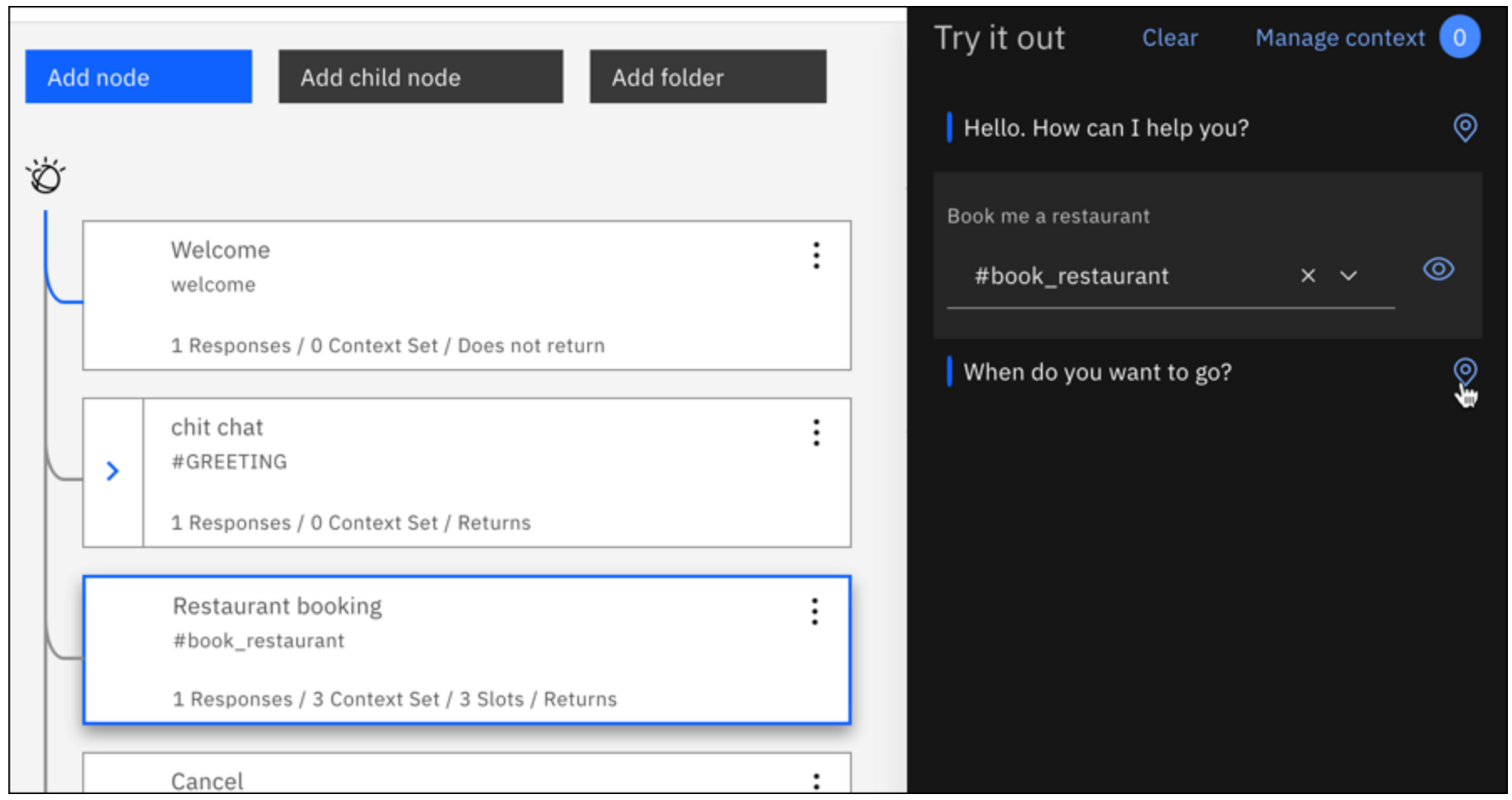
Step 2: Step 2: Temporarily digressing away from dialog

Digressions allow users to break away from a dialog branch to temporarily change the topic before they return to the original dialog flow. In this step, you start to book a restaurant reservation, then digress away to ask for the restaurant's hours. The assistant provides opening hours information, then returns to the restaurant booking dialog flow.

1. Click **Dialog** to open the dialog tree.
2. Click **Try it**.
3. Type `Book me a restaurant`.

   Your assistant responds with `When do you want to go?`.
4. To highlight the node that triggered the response, click the **Location** icon .

   The **Restaurant booking** node is highlighted in the dialog tree.



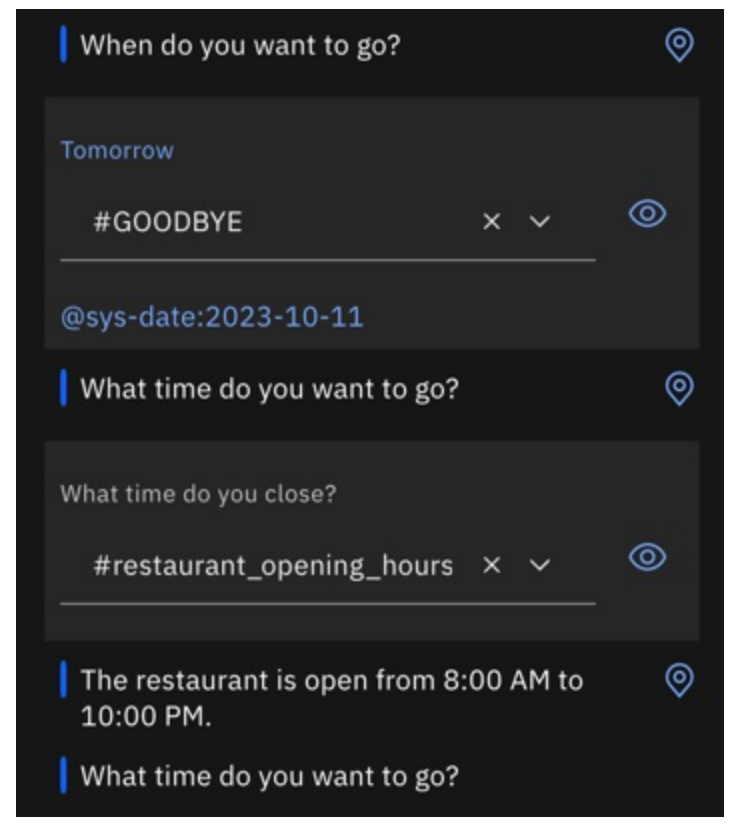
Restaurant booking node

5. Type `Tomorrow`.

   Your assistant responds with a prompt for the time to reserve `What time do you want to go?`.

6. You do not know when the restaurant closes, so you ask, `What time do you close?`

   The assistant digresses away from the restaurant booking node to process the **Restaurant opening hours** node. It responds with, `The restaurant is open from 8:00 AM to 10:00 PM.`. Your assistant then returns to the restaurant booking node, and prompts you again for the reservation time.



Digression

7. **Optional**: To complete the dialog flow, type `8pm` for the reservation time and `2` for the number of guests.

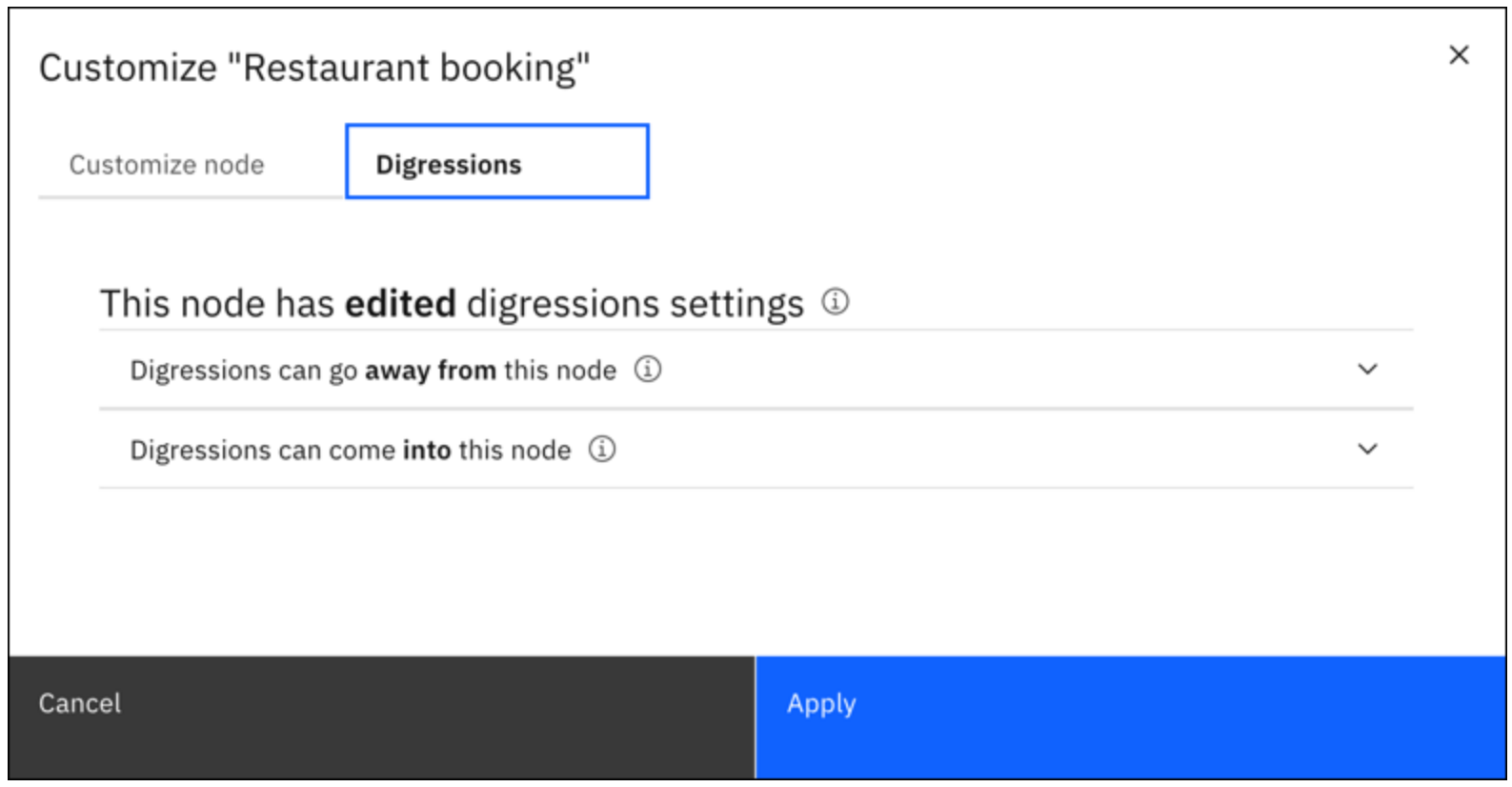
You digressed away from and returned to a dialog flow.

Step 3: Step 3: Disabling slot digressions

In this step, you edit the digression setting for the restaurant booking node to prevent users from digressing away from it, and see how the setting change impacts the dialog flow.

Look at the current digression settings for the **Restaurant booking** node.

1. Click the node to open it in edit view.
2. Click **Customize**, and then click the **Digressions** tab.



Digression settings

3. Click to expand **Digressions can go away from this node** .
4. Set the **Allow digressions away while slot filling** switch to **Off**, and then click **Apply**.
5. Click the **Close** icon to close the node edit view.
6. Click **Try it**.
7. Click **Clear** in the "Try it out" pane to start over.
8. Type `Book me a restaurant`.

   Your assistant responds with a prompt for the day to reserve, `When do you want to go?`
9. Type `Tomorrow`.

   Your assistant responds with a prompt for the time to reserve, `What time do you want to go?`
10. Ask, `What time do you close?`

    Your assistant recognizes that the question triggers the `#restaurant_opening_hours` intent, but ignores it and displays the prompt that is associated with the `@sys-time` slot again instead.

You prevented the user from digressing away from the restaurant booking process.

Step 4: Step 4: Digressing to a node that does not return

You can configure a dialog node to not go back to the node that your assistant digressed away from for the current node to be processed. To demonstrate this configuration, you change the digression setting for the restaurant hours node. In Step 2, after you digress from the restaurant booking node to go to the restaurant opening hours node, your assistant returns to the restaurant booking node to continue with the reservation. In this exercise, after you change the setting, you digress from the **Job opportunities** node to ask for opening hours and see that your assistant does not return to where it left off.

1. Click to open the **Restaurant opening hours** node.
2. Click **Customize**, and then click the **Digressions** tab.
3. Click to expand **Digressions can come into this node** , and deselect the **Return after digression** checkbox.
4. Click **Apply**.
5. Click the **Close** icon to close the node edit view.
6. Click **Try it**.
7. Click **Clear** in the "Try it out" pane to start over.
8. Type `I'm looking for a job`.

   Your assistant responds by saying, `We are always looking for talented people to add to our team. What type of job are you interested in?`
9. Instead of answering this question, ask an unrelated question. Type `What time do you open?`

Your assistant digresses away from the *Job opportunities* node to the *Restaurant opening hours* node to answer your question. Your assistant responds with `The restaurant is open from 8:00 AM to 10:00 PM.`

Unlike in the previous test, this time the dialog does not pick up where it left off in the **Job opportunities** node. Your assistant does not return to the dialog that was in progress because you changed the setting on the **Restaurant opening hours** node to not return.

You digressed away from a dialog without returning.

Summary

In this tutorial you experienced how digressions work, and saw how individual dialog node settings can impact the digressions behavior.

Dialog development

Anatomy of a message

A single `/message` API call is equivalent to a single turn in a conversation, which consists of a message that is submitted by a customer and a corresponding response from your assistant.

Each reply that a customer makes in response to a prompt from the assistant is passed as an independent `/message` API call.

The body of the `/message` API call request and response includes the following objects:

- `context`: Contains variables that are meant to be persisted. For the dialog to reference information that is submitted by the user, you must store the information in the context object. For example, the dialog can collect the user's name and then refer to the user by name in subsequent nodes. The following example shows how the context object is represented in the dialog JSON editor:

  ```
  {
    "context" : {
      "user_name" : "<? @name.literal ?>"
    }
  ```

  For more information, see Retaining information across dialog turns.

- `input`: The string of text that was submitted by the user. The text string can contain up to 2,048 characters. The following example shows how the `input` object is represented in the dialog JSON editor:

  ```
  {
    "input" : {
      "text" : "Where's your nearest store?"
    }
  ```

- `output`: The dialog response to return to the user. The following example shows how the output object is represented in the dialog JSON editor:

  ```
  {
  "output": {
    "generic": [
      {
        "values": [
          {
            "text": "This is my response text."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
  }
  ```

In the resulting API `/message` response, the text response is formatted as follows:

```
$ {
   "text": "This is my response text.",
   "response_type": "text"
}
```

You can define more response types other than a text response. See Responses for more details.

For more information about the `/message` API call, see the API reference.

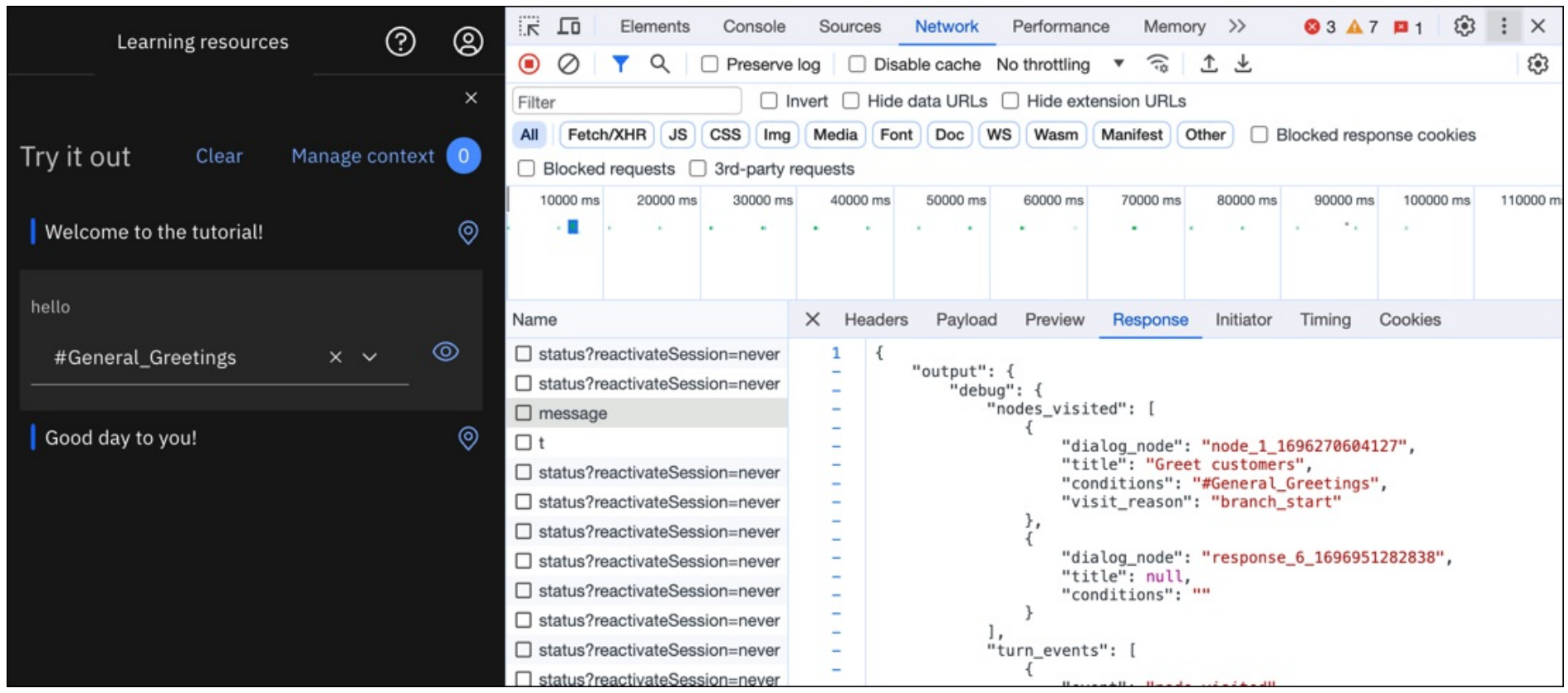
For information about how to refer to these message objects within a conversation, see Expressions for accessing objects.

Viewing API call details

As you test your conversation, you might want to know what the underlying API calls look like that are being returned from the service. You can use the developer tools that are provided by your web browser to inspect them.

From Chrome, for example:

1. Open the developer tools.
2. Open the Network tool. The **Name** section lists multiple API calls.
3. Click the message call that is associated with your test utterance
4. Click the **Response** column to see the API response body. It lists the intents and entities that were recognized in the user input with their confidence scores. It also lists the values of context variables at the time of the call.



Use Chrome developer tools to see API response

1. To view the response body in structured format, click the **Preview** column.

Adding custom dialog flows for integrations

Use the JSON editor in dialog to access information that is submitted from the web chat integration.

Starting with API version `2020-04-01`, the `context` object that is passed as part of the v2 `/message` API request contains an `integrations` object. This object makes it possible to pass information that is specific to a single integration type in the context. For more information, see Context variables.

> ⚠ **Important:** The `integrations` object is available in the v2 API in version `2020-04-01` or later only.

To take advantage of the `context.integrations` object, you can create context variables that are named as follows to get and set values for different integrations:

Integration type	Context variable syntax
Phone	`$integrations.voice_telephony`
Salesforce service desk from web chat	`$integrations.salesforce`
SMS with Twilio	`$integrations.text_messaging`
Web chat (and assistant preview)	`$integrations.chat`

Zendesk service desk from web chat	`$integrations.zendesk`

Integration-specific context variables

Building integration-specific dialog flow

Create a single dialog that is optimized to use the best features that are offered by each channel or client interface in which it is deployed.

You can customize the conversation in the following ways:

- To add an entire dialog branch that is only processed by a specific integration type, add the appropriate integration type context variable, such as `$integrations.chat`, to the *If assistant recognizes* field of the dialog root node.
- To add a single dialog node that is only processed by a specific integration type, add the appropriate integration type context variable, such as `$integrations.zendesk`, to the *If assistant recognizes* field of the dialog child node.
- To add slightly different responses for a single dialog node based on the integration type, complete the following steps:
  - From the node's edit view, click **Customize** and then set the *Multiple conditioned responses* switch to **On**. Click **Apply**.
  - In the dialog node response section, add the appropriate condition and corresponding response for each custom response type.

    The following examples show how to specify a hypertext link in the best format for the integration where the text response is displayed. For the *Web chat* integration, which supports Markdown formatting, you can include a link label in the response text to make the response look nicer. For the *SMS with Twilio* integration, you can skip the formatting that makes sense in a web page, and add the straight URL.

Integration type	Condition	Sample text response
SMS with Twilio	`$integrations.text_messaging`	`For more information, go to https://www.ibm.com.`
Web chat	`$integrations.chat`	`For more information, go to [the ibm.com site] (https://www.ibm.com).`
Response to show if no other conditions are met.	`true`	`For more information, go to ibm.com.`

Custom conditioned responses

The rich response types often behave differently when they are displayed in different built-in integrations or in the assistant preview.

Note: If you need to provide customized responses for different channels, and you do not need to modify your dialog flow based on which integration is in use, you can also use the `channels` array to target your responses to specific integrations.

Web chat: Accessing sensitive data

If you enable security for the web chat, you can configure your web chat implementation to send encrypted data to the dialog. Payload data that is sent from web chat is stored in a private context variable named `context.integrations.chat.private.user_payload`. No private variables are sent from the dialog to any integrations. For more information about how to pass data, see Encrypting sensitive data in web chat.

To access the payload data, you can reference the `context.integrations.chat.private.user_payload` object from the dialog node condition.

Note: You must know the structure of the JSON object that is sent in the payload.

For example, if you passed the value `"mvp:true"` in the JSON payload, you can add a dialog flow that checks for this value to define a response that is meant for VIP customers only. Add a dialog node with a condition like this:

Field	Value
If assistant recognizes	`$integrations.chat.private.user_payload.mvp`
Assistant responds	`I can help you reserve box seats at the upcoming conference!`

Private variable as node condition

Web chat: Accessing web browser information

When you use the web chat integration, information about the web browser that your customer is using to access the web chat is automatically collected and stored. The information is stored in the `context.integrations.chat.browser_info` object.

You can design your dialog to take advantage of details about the web browser in use. The following properties are taken from the `window` object that represents the window in which the web chat is running:

- `browser_name`: The browser name, such as `chrome`, `edge`, or `firefox`.
- `browser_version`: The browser version, such as `80.0.0`.
- `browser_OS`: The operating system of the customer's computer, such as `Mac OS`.
- `language`: The default locale code of the browser, such as `en-US`.
- `page_url`: Full URL of the web page in which the web chat is embedded. For example,: `https://www.example.com/products`
- `screen_resolution`: Specifies the height and width of the browser window in which the web page is displayed. For example,: `width: 1440, height: 900`
- `user_agent`: Content from the User-Agent request header. For example,: `Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:80.0) Gecko/20100101 Firefox/80.0`
- `client_ip_address`: IP address of the customer's computer. For example,: `183.49.92.42`

Example: Using page URL information in your dialog

You might embed the web chat in multiple pages on your website. Perhaps you want to route chat transfers from the web chat to different Salesforce agents based on the page from which the customer asks to speak to someone. If the customer is on the insurance plans page of your website (`https://www.example.com/insurance.html`), you want to route them to agents who are experts in your company's insurance plans. If the customer is on the investments web page (`https://www.example.com/invest.html`), you want to route them to agents who are experts in your company's investment opportunities. You can add multiple conditioned responses to the dialog node where the transfer takes place and add logic like this:

Conditioned response 1

- Condition: `$integrations.chat.browser_info.page_url.contains('insurance.html')`
- Response type: *Connect to human agent*
- Button ID: `Z23453e25vv` (The button that routes to the insurance experts agent queue)

Conditioned response 2

- Condition: `$integrations.chat.browser_info.page_url.contains('invest.html')`
- Response type: *Connect to human agent*
- Button ID: `Z23453j24ty` (The button that routes to the investment experts agent queue)

Web chat: Reusing the JWT for webhook authentication

You can use the same JSON Web Token (JWT) that was used to secure your web chat to authenticate webhook calls. If you specify a token in the `identityToken` property when you add the web chat to your web page, the token is stored in a private variable named `context.integrations.chat.private.jwt`. For more information about passing a JWT, see Enabling web chat security.

1. In your dialog, open **Webhooks**.
2. Click **Add header**.
3. In **Header name**, enter any name, such as `JWT`.
4. In the **Header value** field, add `$integrations.chat.private.jwt`.

Handling SMS with Twilio interactions

Learn about common tasks that you can use to manage the flow of conversations that your assistant has with customers by using SMS text messaging.

Before you add customizations to your dialog that support SMS messaging interactions, you must set up the SMS with Twilio integration. For more information, see Integrating with SMS with Twilio.

You can perform the following types of tasks:

- Sending multimedia content over text
- Customizing lists in a text message
- Sending a text message during a phone conversation

For command reference documentation, see SMS integration reference.

Adding SMS-based actions to your dialog

When you call messaging-specific actions from a dialog, follow these guidelines:

- Define the SMS action within the `context` object, not the `output` object of the dialog node JSON snippet.
- Define only one SMS action or one sequence per conversation turn.
- Do not jump from a dialog node with an SMS action that is configured for it to another dialog node with an action configured for it.

To enable text messaging-specific actions, you must add a JSON code block to the dialog node where you want the SMS action to trigger.

To add a JSON code block to a dialog node, complete the following steps:

1. Click **Dialog** to open the dialog tree.
2. Open the dialog node where you want to call the action.
3. In **Assistant responds**, click the options menu ⋮ menu, and then select **Open JSON editor**.
4. Add the `smsAction` command JSON code block to the `context` object. (If no `context` object exists, add one. The `context` object is a peer to the `output` object.)

   For example,:

```
{
  "output": {
    "generic": [
    ]
  },
  "context": {
    "smsAction": {
      "command": "<command-name>",
      "parameters": {
        "<first-parameter>": "<parameter-value>"
      }
    }
  }
}
```

Sending multimedia content over text

To allow multimedia content, such as an image, to be sent in a text message, use the `smsActSendMedia` command.

```
{
  "output": {
      "generic": [
      ]
  },
  "context": {
    "smsAction": {
        "command": "smsActSendMedia",
        "parameters":{
          "mediaURL": [
            "https://example.com/images/image.png"
          ]
        }
    }
  }
}
```

You can specify the following parameter value for the `vsmsActSendMedia` command:

- `mediaURL`: A JSON array of one or more publicly-accessible media URLs for images or videos.

Customizing lists

You can customize how these lists are displayed and handled.

- Options list
- Disambiguation

Options list

The dialog supports an `option` response type, which shows the customer multiple options to choose from. You can customize how the options that are defined for an option response type are shown to customers and the ways in which a customer can select an option by adding the `vgwActSetOptionsConfig` action command.

The following example shows how to customize the option response type.

```
{
  "output": {
    "generic": [
      {
      "title": "Which of these items do you want to insure?",
      "options": [
      {
        "label": "Boat",
        "value": {
          "input": {
            "text": "I want to buy boat insurance."
          }
        },
          "label": "Car",
          "value": {
            "input": {
              "text": "I want to buy car insurance."
            }
          },
          "label": "House",
          "value": {
            "input": {
              "text": "I want to buy house insurance."
            }
          }
        }
      ],
      "description": "Insurance types.",
      "response_type": "option"
      }
  ]
},
"context": {
  "smsAction": {
    "command": "smsActSetOptionsConfig",
    "parameters": {
      "prefixText": "%s."
    }
  }
}
```

First, the value that is specified in the `title` attribute is displayed to the user. Then, the text specified in each `label` attribute. For example, `Which of these items do you want to insure? 1.Boat 2.Car 3.House`

To configure what the assistant shows before each option, edit the `prefixText` parameter. Use `%s` to represent the number corresponding to the option; it is replaced with the actual number at run time.

```
"smsAction": {
  "command": "smsActSetOptionsConfig",
  "parameters": {
    "prefixText": "Enter %s for "
  }
}
```

For example, `Which of these items do you want to insure? Enter 1 for Boat Enter 2 for Car Enter 3 for House`

Disambiguation list

When the dialog is confident that more than one dialog node might be the right one to process in response to a customer query, disambiguation is triggered. Disambiguation asks the customer to clarify which path they want to follow to get an answer. For more information, see Disambiguation.

You can customize how the disambiguation list choices are displayed and how a customer can select a disambiguation choice by adding the `smsActSetDisambiguationConfig` action command.

You might want to define the customization in the welcome node or another node that is triggered early in the conversation so that it is applied any time disambiguation is triggered.

```
{
  "output": {
     "generic": [
     ]
  },
  "context": {
    "smsAction": {
      "command": "smsActSetDisambiguationConfig",
      "parameters": {
        "prefixText": "%s."
      }
    }
  }
}
```

When displayed, the assistant shows the introductory text that is configured for disambiguation, such as `Did you mean?`. Then, it lists the choices from the disambiguation list as numbered choices.

The `prefixText` parameter adds a number prefix to the text specified in a `label`. The list choices are numbered sequentially and are displayed to the user in the order in which they appear in the list. The user can type a number to pick one of the choices.

For example, if label is configured as follows:

```
"label": "I'd like to order a drink."
```

The assistant sends this message to the user:

```
1. I'd like to order a drink.
```

You can configure the text that is prepended to each label. In the `prefixText` attribute, `%s` represents the number corresponding to the suggestion; it is replaced with the actual number at run time.

```
"context": {
   "smsAction": {
     "command": "smsActSetDisambiguationConfig",
     "parameters": {
       "prefixText": "Enter %s for:"
     }
   }
 }
```

When `prefixText` is set to `Enter %s for:`, the following output is sent to the customer:

```
Enter 1 for: I'd like to order a drink.
```

Requesting client actions

Add a client action to your dialog node to request that a client-side process returns a result to the dialog.

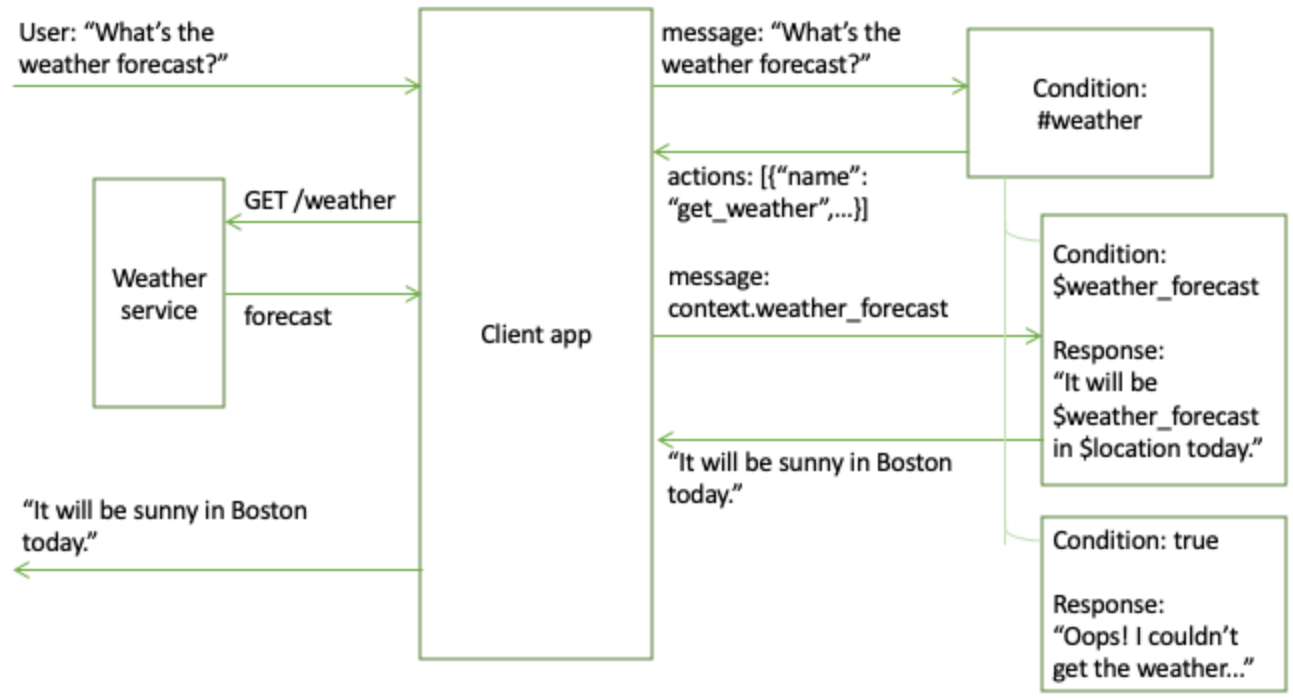
If your assistant is integrated with a custom client that uses the API, you can use client actions to start functions that the client application completes. A client action defines a request for a client-side process, in a standardized format that your external client application can understand. Your external client application must use the provided information to run the requested action, which can be a local programmatic function, a call to an external service, or any other action the client can do. The client then returns the result from the action to the dialog, which can process it further, display it to the user, or use it as a condition to determine subsequent flow.

Unlike other action types, a client action does not make a direct call itself. Instead, it makes a request for the client application to do something, in the form of an action object that is included in a response from the dialog (along with any output text). This object includes a name that identifies the requested action, along with any required parameters and the name of a context variable where the result is stored. It is the responsibility of the client application to do the requested action, store the result in the context, and send it back to the dialog with the next message.

You might call a client application to do the following types of things:

- Validate information that you collected from the user.
- Do calculations or string manipulations on user input that are too complex for supported SpEL expression methods to handle.
- Get data from another application or service.

The following diagram illustrates how client actions work.



Client action example

The flow of requests and responses follows this pattern:

1. The client application sends a message that contains user input that asks for a weather forecast (by using the `message` or `message_stateless` method).
2. The user input triggers a dialog node that is conditioned on a #weather intent. In its response to the client, this node specifies the `get_weather` client action, which is a name that the client application recognizes. (This is in addition to any text response, such as `Checking the weather forecast...`.)
3. When it receives this response, the client application recognizes that the `get_weather` action is being requested. It calls an external web service (`/weather`) to get the actual forecast information, passing any specified parameters (such as the user's location). The client application then stores the returned information in the context variable that is specified in the action request (`$weather_forecast`).
4. The client application sends another message to the service, including the updated context that contains the weather forecast information. From the dialog's perspective, this message is effectively the next round of user input, although the actual input text is blank.
5. A child node of the #weather node is triggered by the presence of the `$weather_forecast` context variable. This node responds with text output that includes the weather forecast, which the client application displays to the user. (If the `$weather_forecast` variable is not set, another child node can handle this case and report an error.)

Tip: It is also possible to call an external web service directly from a dialog node, without involving the client application, by defining a webhook. For more information about how to call an external service by using a webhook, see Making a programmatic call from a dialog node.

Procedure

To request a client action from a dialog node, complete the following steps:

1. Click **Dialog** to open the dialog tree.
2. Open the dialog node where you want to call the action.
3. In **Assistant responds**, click the options menu ⋮ menu, and then select **Open JSON editor**.
4. Use the following syntax to define the client action you want to request.

```
{
  "context": {
    "variable_name" : "variable_value"
  },
  "actions": [
    {
```

```
      "name":"<actionName>",
      "type":"client",
      "parameters": {
        "<parameter_name>":"<parameter_value>",
        "<parameter_name>":"<parameter_value>"
      },
      "result_variable": "<result_variable_name>"
    }
  ],
  "output": {
    "text": "response text"
  }
}
```

The `actions` array specifies the actions that you want the client application to do. It can define up to five separate actions. Specify the following name and value pairs in the JSON array:

- `<actionName>`: Required. The name of the action you want the client application to do. This name can be in any format (for example, `calculateRate` or `get_balance`), but it must be a name that your client application recognizes and knows how to handle. The name cannot be longer than 256 characters.
- `<type>`: Indicates the type of call to make. For a client action, this property is optional (`client` is the default value).
- `<action_parameters>`: Any parameters that are required for the client action, which are specified as a JSON object. If the action does not require any parameters, omit this property.
- `<result_variable_name>`: The name of the context variable you want to use to store the JSON object that is returned by the client action. The client application is expected to use the specified variable to send the return value in the context of the next `/message` input, and subsequent dialog nodes can then access it. You can specify the `result_variable_name` by using the following syntax:
  - `my_result`
  - `$my_result`

  The name cannot be longer than 64 characters. The variable name cannot contain the following characters: parentheses `()`, brackets (`[]`), a single quotation mark (`'`), a quotation mark (`"`), or a backslash (`\`).

  You can optionally specify a `context.` location keyword prefix for this variable (for example, `context.my_result`). However, this is not necessary because all context variables are stored in this location by default.

  You can include periods in the variable name to create a nested JSON object. For example, you can define these variables to capture results from two separate requests to a weather service for forecasts for today and tomorrow:

  - `context.weather.today`
  - `context.weather.tomorrow`

  The results (in this example, `temp` and `rain` properties) are stored in the context in this structure:

```
{
  "weather": {
    "today": {
      "temp": "20",
      "rain": "30"
    },
    "tomorrow": {
      "temp": "23",
      "rain": "80"
    }
  }
}
```

  If multiple actions in a single JSON action array add the result of their programmatic call to the same context variable, then the order in which the context is updated matters. Per action type, the order definition of the actions determines the order in which the context variable's value is set. The context variable value that is returned by the last action in the array overwrites the values that are calculated by any other actions.

  The `result_variable` property is required. If the client action does not return any result, specify `null` as the value.

Client action example

The following example shows what a request for a call to an external weather service might look like. It is added to the JSON editor that is associated with

the node response. Slots collect and store the date and location information from the user by the time that the node-level response is triggered. This example assumes that the client action is named `weather_forecast`, that it takes a `location` parameter, and that the results are to be stored in the `weather_forecast` context variable.

```
{
 "actions": [
  {
   "name": "get_weather",
   "type": "client",
   "parameters": {
    "location": "$location"
   },
   "result_variable": "weather_forecast"
  }
 ]
}
```

The client application must check for the presence of any client actions in the responses to messages it sends to the assistant. When it recognizes a request for the `get_weather` action, it runs the action (calling the external weather service), and it stores the result in the specified context variable (`weather_forecast`). It then sends a message to the service, including the updated context.

To handle this message in your dialog, create a child node after the node that requested the action. You can condition this child node on the special condition `true` to ensure that it is always triggered by the message that the client sends after it completes the requested action. In this child node, add the response to show the user, reading the stored action result from the `$my_forecast` context variable:

```
{
  "output": {
    "text": {
      "values": [
        "It will be $weather_forecast in $location.literal today."
      ]
    }
  }
```

Modifying a dialog by using the API

The REST API supports modifying your dialog programmatically. You can use the `/dialog_nodes` API to create, delete, or modify dialog nodes.

A dialog is a tree of interconnected nodes, and that it must conform to certain rules to be valid. Any change that you make to a dialog node might have cascading effects on other nodes, or on the structure of your dialog. Before you use the `/dialog_nodes` API to modify your dialog, make sure you understand how your changes affect the rest of the dialog. You can make a backup copy of the current dialog For more information, see Backing up and restoring data.

A valid dialog always satisfies the following criteria:

- Each dialog node has a unique ID (the `dialog_node` property).
- A child node is aware of its parent node (the `parent` property). However, a parent node is not aware of its children.
- A node is aware of its immediate previous sibling, if any (the `previous_sibling` property). All siblings that share the parent form a linked list, with each node pointing to the previous node.
- Only one child of a parent can be the first sibling (meaning that its `previous_sibling` is null).
- A node cannot point to a previous sibling that is a child of a different parent.
- Two nodes cannot point to the same previous sibling.
- A node can specify another node that is to be run next (the `next_step` property).
- A node cannot be its own parent or its own sibling.
- A node must have a type property that contains one of the following values. If no type property is specified, then the type is `standard`.
    - `event_handler`: A handler that is defined for a frame node or an individual slot node.

    From the tool, you can define a frame node handler by clicking the **Manage handlers** link from a node with slots. (The tool user interface does not expose the slot-level event handler, but you can define one through the API.)

    - `frame`: A node with one or more child nodes of type `slot`. Any child slot nodes that are required must be filled before the service can exit the frame node.

The frame node type is represented as a node with slots in the tool. The node that contains the slots is represented as a node of type= `frame`. It is the parent node to each slot, which is represented as a child node of type `slot`.

  - `response_condition`: A conditional response.

  In the tool, you can add one or more conditional responses to a node. Each conditional response that you define is represented in the underlying JSON as an individual node of type= `response_condition`.

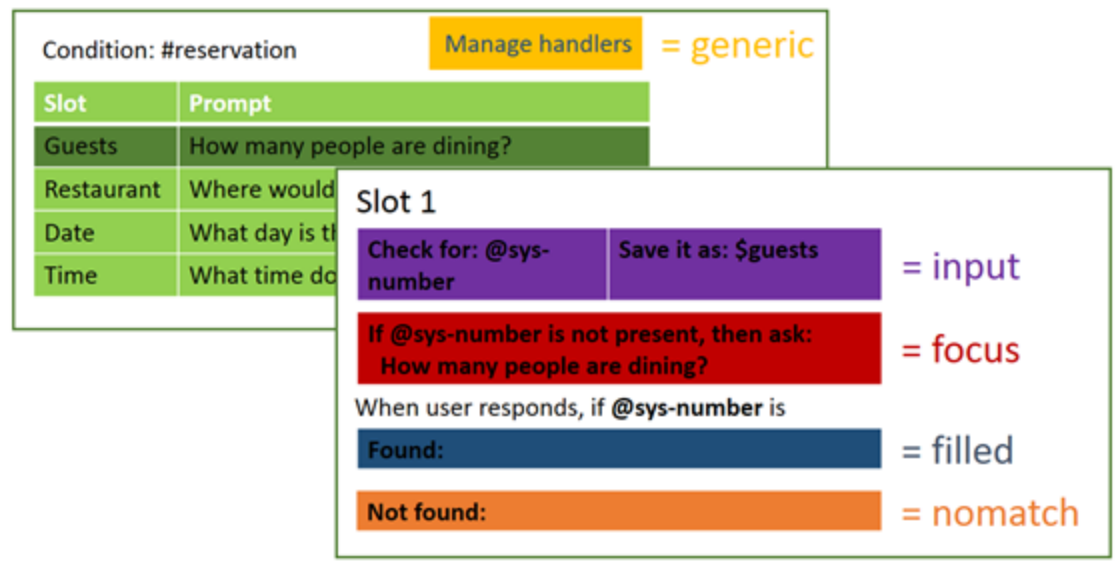
  - `slot`: A child node of a node of type `frame`.

  This node type is represented in the tool as being one of multiple slots that are added to a single node. That single node is represented in the JSON as a parent node of type `frame`.

  - `standard`: A typical dialog node. This is the default type.

- For nodes of type `slot` that have the same parent node, the sibling order (specified by the `previous_sibling` property) reflects the order in which the slots are processed.
- A node of type `slot` must have a parent node of type `frame`.
- A node of type `frame` must have at least one child node of type `slot`.
- A node of type `response_condition` must have a parent node of type `standard` or `frame`.
- Nodes of type `response_condition` and `event_handler` cannot have children.
- A node of type `event_handler` must also have an `event_name` property that contains one of the following values to identify the type of node event:
  - `filled`: Defines what to do if the user provides a value that meets the condition that is specified in the *Check for* field of a slot, and the slot is filled. A handler with this name is only present if a Found condition is defined for the slot.
  - `focus`: Defines the question to show that prompts the user to provide the information needed by the slot. A handler with this name is only present if the slot is required.
  - `generic`: Defines a condition to watch for that can address unrelated questions that users might ask while filling a slot or node with slots.
  - `input`: Updates the message context to include a context variable with the value that is collected from the user to fill the slot. A handler with this name must be present for each slot in the frame node.
  - `nomatch`: Defines what to do if the user's response to the slot prompt does not contain a valid value. A handler with this name is only present if a Not found condition is defined for the slot.

  The following diagram illustrates where in the tool user interface that you define the code that is triggered for each named event.



Event handlers

- A node of type `event_handler` with an event_name of `generic` can have a parent of type `slot` or `frame`.
- A node of type `event_handler` with an event_name of `focus`, `input`, `filled`, or `nomatch` must have a parent of type `slot`.
- If more than one event_handler with the same event_name is associated with the same parent node, then the order of the siblings is the order in which the event handlers are run.
- For `event_handler` nodes with the same parent slot node, the order of execution is the same regardless of the placement of the node definitions. The events are triggered in this order by event_name:
  1. focus
  2. input

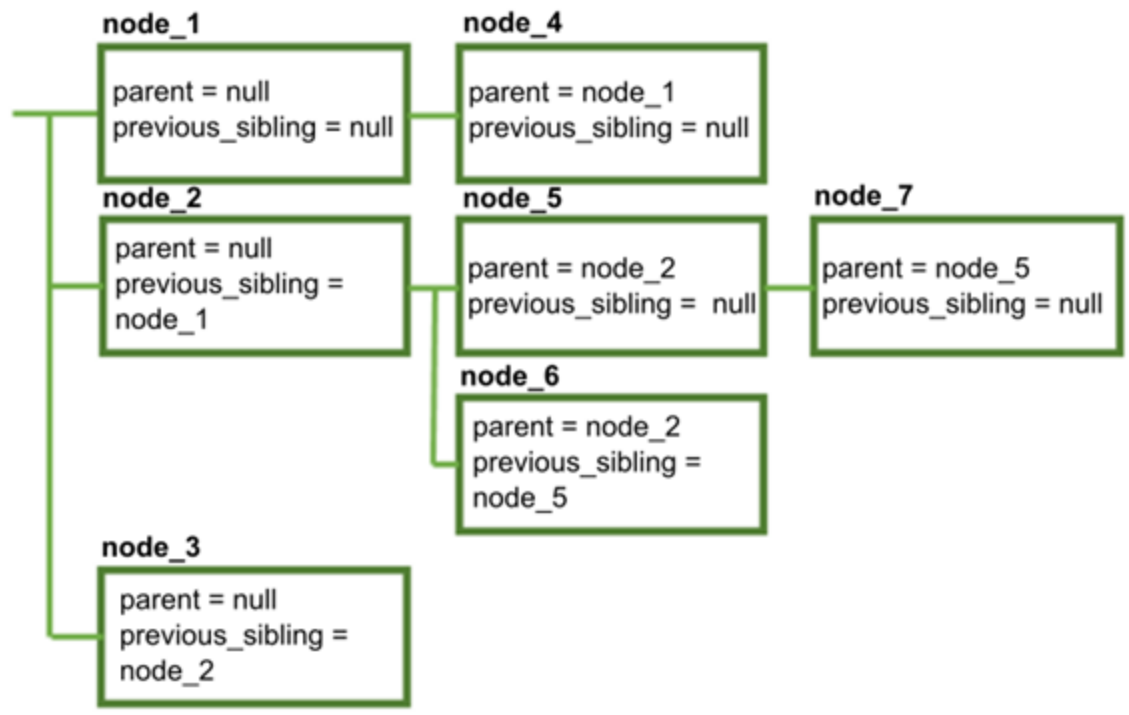
3. filled
4. generic*
5. nomatch

*If an `event_handler` with the event_name `generic` is defined for this slot or for the parent frame, then it is run between the filled and nomatch event_handler nodes.

The following examples show how various modifications might cause cascading changes.

Creating a node

Consider the following simple dialog tree:

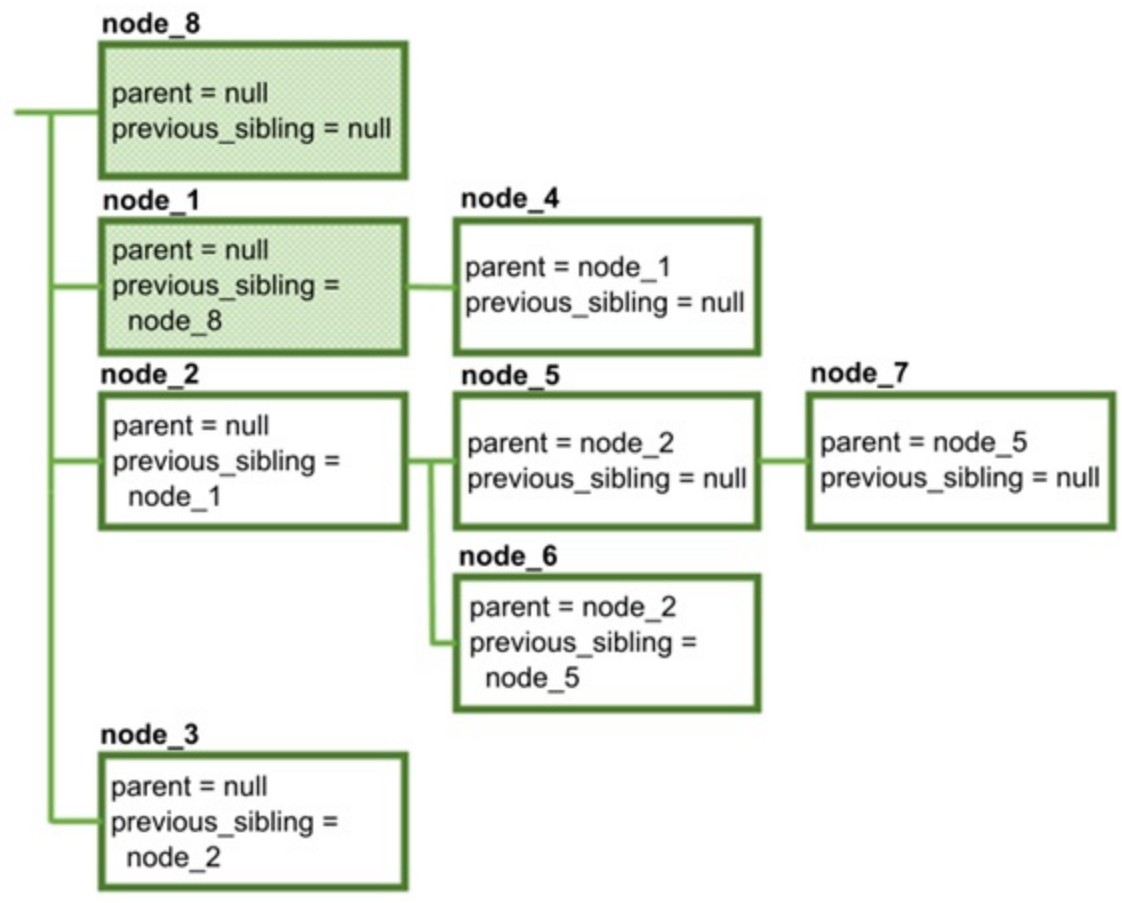


Example dialog

We can create a new node by making a POST request to /dialog_nodes with the following body:

```
$ {
  "dialog_node": "node_8"
}
```

The dialog now looks like this:



Example dialog 2

Because **node_8** was created without specifying a value for `parent` or `previous_sibling`, it is now the first node in the dialog. In addition to creating **node_8**, the service also modified **node_1** so that its `previous_sibling` property points to the new node.

You can create a node somewhere else in the dialog by specifying the parent and previous sibling:

```
{
  "dialog_node": "node_9",
  "parent": "node_2",
  "previous_sibling": "node_5"
}
```

The values that you specify for `parent` and `previous_node` must be valid:

- Both values must refer to existing nodes.
- The specified parent must be the same as the parent of the previous sibling (or `null`, if the previous sibling has no parent).
- The parent cannot be a node of type `response_condition` or `event_handler`.

The resulting dialog looks like this:

Example dialog 3

In addition to creating **node_9**, the service automatically updates the `previous_sibling` property of *node_6* so that it points to the new node.

Moving a node to a different parent

Move **node_5** to a different parent by using the POST `/dialog_nodes/node_5` method with the following body:

```
{
  "parent": "node_1"
}
```

The specified value for `parent` must be valid:

- It must refer to an existing node.
- It must not refer to the node that is modified (a node cannot be its own parent).
- It must not refer to a descendant of the node that is modified.
- It must not refer to a node of type `response_condition` or `event_handler`.

This results in the following changed structure:

Example dialog 4

Several things happened here:

- When **node_5** moved to its new parent, **node_7** went with it (because the `parent` value for **node_7** did not change). When you move a node, all descendants of that node stay with it.
- Because we did not specify a `previous_sibling` value for **node_5**, it is now the first sibling under **node_1**.
- The `previous_sibling` property of **node_4** was updated to `node_5`.
- The `previous_sibling` property of **node_9** was updated to `null` because it is now the first sibling under **node_2**.

Resequencing siblings

Now set **node_5** as the second sibling instead of the first by using the `POST /dialog_nodes/node_5` method with the following body:

```
{
  "previous_sibling": "node_4"
}
```

When you modify `previous_sibling`, the new value must be valid:

- It must refer to an existing node
- It must not refer to the node that is modified (a node cannot be its own sibling)
- It must refer to a child of the same parent (all siblings must have the same parent)

The structure changes as follows:

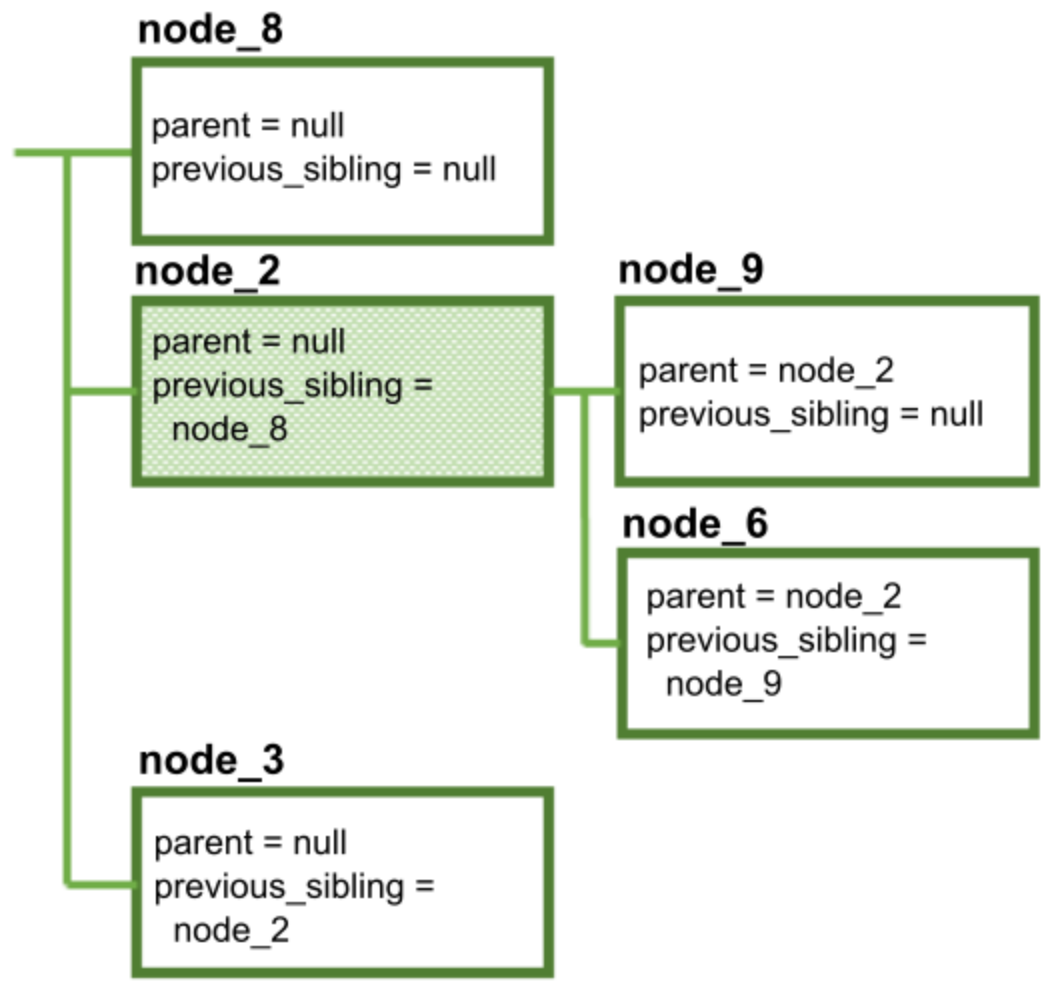
Example dialog 5

Node_7 stays with its parent. In addition, **node_4** is modified so that its `previous_sibling` is `null` because it is now the first sibling.

Deleting a node

Delete **node_1** by using the `DELETE /dialog_nodes/node_1` method.

The result is:



Example dialog 6

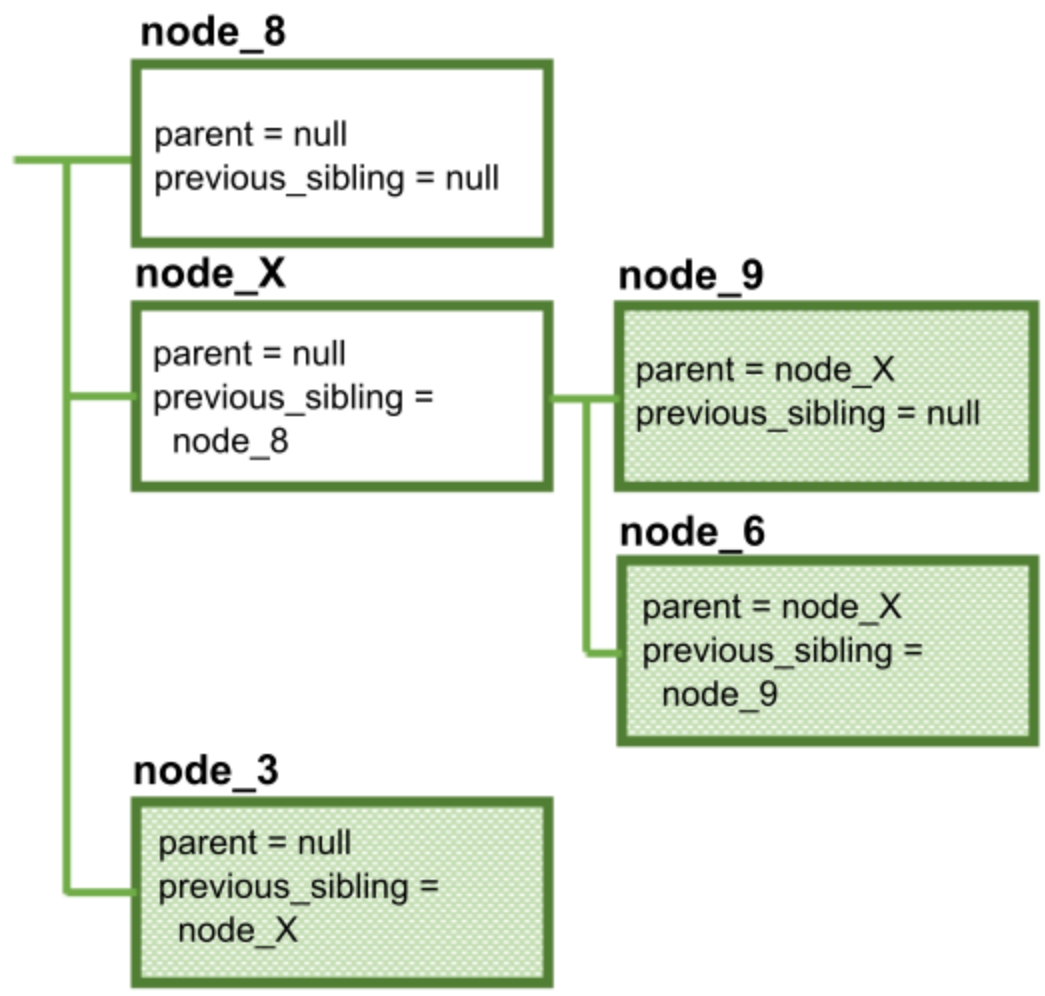
Node_1, **node_4**, **node_5**, and **node_7** were all deleted. When you delete a node, all descendants of that node are deleted as well. Therefore, if you delete a root node, you are deleting an entire branch of the dialog tree. Any other references to the deleted node (such as `next_step` references) are changed to `null`.

In addition, **node_2** is updated to point to **node_8** as its new previous sibling.

Renaming a node

Rename **node_2** using the `POST /dialog_nodes/node_2` method with the following body:

```
{
  "dialog_node": "node_X"
}
```



Example dialog 7

The structure of the dialog didn't change, but multiple nodes were modified to reflect the changed name:

- The `parent` properties of **node_9** and **node_6**
- The `previous_sibling` property of **node_3**

Any other references to the deleted node (such as `next_step` references) are also changed.

Installing on IBM On-premises

IBM Software Hub

Installing watsonx Assistant for IBM Software Hub

Learn how to install watsonx Assistant for IBM Software Hub.

IBM® Software Hub is a cloud-native solution that enables you to install, manage, and monitor IBM solutions on Red Hat OpenShift Container Platform.

IBM Software Hub also includes a centralized management interface, called IBM Software Hub Control Center. With Control Center you can manage multiple instances of IBM Software Hub on the same cluster. IBM Software Hub provides an integrated web client that makes it easy for users to move between any solutions that are installed on the same instance.

The installation process differs depending on the version you are installing. The following table shows the available versions.

Version	Cluster	Installation instructions
5.2.1	IBM Software Hub 5.2.x	Installing 5.2.1
5.2.0	IBM Software Hub 5.2.x	Installing 5.2.0
5.1.3	IBM Software Hub 5.1.x	Installing 5.1.3
5.1.2	IBM Software Hub 5.1.x	Installing 5.1.2
5.1.1	IBM Software Hub 5.1.x	Installing 5.1.1
5.1.0	IBM Software Hub 5.1.x	Installing 5.1.0

Available versions

Support matrix

The following table describes which versions of watsonx Assistant are supported on which versions of IBM Software Hub and Red Hat OpenShift.

watsonx Assistant version	IBM Software Hub version	Red Hat OpenShift version
5.2.1	5.2.x	4.12, 4.14, 4.15, 4.16, 4.17, or 4.18
5.2.0	5.2.x	4.12, 4.14, 4.15, 4.16, 4.17, or 4.18
5.1.3	5.1.x	4.12, 4.14, 4.15, 4.16, 4.17, or 4.18
5.1.2	5.1.x	4.12, 4.14, 4.15, 4.16, or 4.17
5.1.1	5.1.x	4.12, 4.14, 4.15, 4.16, or 4.17
5.1.0	5.1.x	4.12, 4.14, 4.15, 4.16, or 4.17

Support matrix

IBM Cloud Pak for Data

Installing watsonx Assistant for IBM Cloud Pak for Data

To learn how to install latest versions of watsonx Assistant for IBM Software Hub, see Installing Software Hub.

Learn how to install watsonx Assistant for IBM Cloud Pak® for Data.

The IBM Cloud Pak for Data environment is a Kubernetes-based container platform that can help you quickly modernize and automate workloads that are associated with the applications and services you use. You can develop and deploy on your own infrastructure and in your data center which helps to

mitigate risk and minimize vulnerabilities.

The installation process differs depending on the version you are installing. The following table shows the available versions.

Version	Cluster	Installation instructions
5.0.3	IBM Cloud Pak for Data 5.0.x	Installing 5.0.3
5.0.1	IBM Cloud Pak for Data 5.0.x	Installing 5.0.1
5.0.0	IBM Cloud Pak for Data 5.0.x	Installing 5.0.0
4.8.9	IBM Cloud Pak for Data 4.8.x	Installing 4.8.9
4.8.8	IBM Cloud Pak for Data 4.8.x	Installing 4.8.8
4.8.7	IBM Cloud Pak for Data 4.8.x	Installing 4.8.7
4.8.6	IBM Cloud Pak for Data 4.8.x	Installing 4.8.6
4.8.5	IBM Cloud Pak for Data 4.8.x	Installing 4.8.5
4.8.4	IBM Cloud Pak for Data 4.8.x	Installing 4.8.4
4.8.3	IBM Cloud Pak for Data 4.8.x	Installing 4.8.3
4.8.2	IBM Cloud Pak for Data 4.8.x	Installing 4.8.2
4.8.0	IBM Cloud Pak for Data 4.8.x	Installing 4.8.0
4.7.4	IBM Cloud Pak for Data 4.7.x	Installing 4.7.4
4.7.2	IBM Cloud Pak for Data 4.7.x	Installing 4.7.2
4.7.1	IBM Cloud Pak for Data 4.7.x	Installing 4.7.1
4.7.0	IBM Cloud Pak for Data 4.7.x	Installing 4.7.0
4.6.5	IBM Cloud Pak for Data 4.6.x	Installing 4.6.5
4.6.3	IBM Cloud Pak for Data 4.6.x	Installing 4.6.3
4.6.2	IBM Cloud Pak for Data 4.6.x	Installing 4.6.2
4.6.0	IBM Cloud Pak for Data 4.6.x	Installing 4.6.0
4.5.3	IBM Cloud Pak for Data 4.5.x	Installing 4.5.3
4.5.1	IBM Cloud Pak for Data 4.5.x	Installing 4.5.1
4.5.0	IBM Cloud Pak for Data 4.5.x	Installing 4.5.0
4.0.8	IBM Cloud Pak for Data 4.0.x	Installing 4.0.8
4.0.7	IBM Cloud Pak for Data 4.0.x	Installing 4.0.7
4.0.6	IBM Cloud Pak for Data 4.0.x	Installing 4.0.6

4.0.5	IBM Cloud Pak for Data 4.0.x	Installing 4.0.5
4.0.4	IBM Cloud Pak for Data 4.0.x	Installing 4.0.4
4.0.2	IBM Cloud Pak for Data 4.0.x	Installing 4.0.2
4.0.0	IBM Cloud Pak for Data 4.0.x	Installing 4.0.0
1.5.0	IBM Cloud Pak for Data 3.0.1 or 3.5	Installing 1.5.0

Available versions

Support matrix

The following table describes which versions of watsonx Assistant are supported on which versions of IBM Cloud Pak for Data and Red Hat OpenShift.

watsonx Assistant version	IBM Cloud Pak for Data version	Red Hat OpenShift version
5.0.3	5.0.x	4.16
5.0.3	5.0.x	4.15
5.0.3	5.0.x	4.14
5.0.3	5.0.x	4.12
5.0.1	5.0.x	4.15
5.0.1	5.0.x	4.14
5.0.1	5.0.x	4.12
5.0.0	5.0.x	4.15
5.0.0	5.0.x	4.14
5.0.0	5.0.x	4.12
4.8.9	4.8.x	4.18
4.8.9	4.8.x	4.17
4.8.9	4.8.x	4.16
4.8.9	4.8.x	4.15
4.8.9	4.8.x	4.14
4.8.9	4.8.x	4.12
4.8.8	4.8.x	4.16
4.8.8	4.8.x	4.15
4.8.8	4.8.x	4.14
4.8.8	4.8.x	4.12

4.8.7	4.8.x	4.16
4.8.7	4.8.x	4.15
4.8.7	4.8.x	4.14
4.8.7	4.8.x	4.12
4.8.6	4.8.x	4.15
4.8.6	4.8.x	4.14
4.8.6	4.8.x	4.12
4.8.5	4.8.x	4.15
4.8.5	4.8.x	4.14
4.8.5	4.8.x	4.12
4.8.4	4.8.x	4.14
4.8.4	4.8.x	4.12
4.8.3	4.8.x	4.14
4.8.3	4.8.x	4.12
4.8.2	4.8.x	4.14
4.8.2	4.8.x	4.12
4.8.0	4.8.x	4.12
4.7.4	4.7.x	4.12
4.7.4	4.7.x	4.10
4.7.2	4.7.x	4.12
4.7.2	4.7.x	4.10
4.7.1	4.7.x	4.12
4.7.1	4.7.x	4.10
4.7.0	4.7.x	4.12
4.7.0	4.7.x	4.10
4.6.5	4.6.x	4.12
4.6.5	4.6.x	4.10
4.6.3	4.6.x	4.10

4.6.2	4.6.x	4.10
4.6.2	4.6.x	4.8
4.6.0	4.6.x	4.10
4.6.0	4.6.x	4.8
4.5.3	4.5.x	4.10
4.5.3	4.5.x	4.8
4.5.3	4.5.x	4.6
4.5.1	4.5.x	4.10
4.5.1	4.5.x	4.8
4.5.1	4.5.x	4.6
4.5.0	4.5.x	4.10
4.5.0	4.5.x	4.8
4.5.0	4.5.x	4.6
4.0.8	4.0.x	4.8
4.0.8	4.0.x	4.6
4.0.7	4.0.x	4.8
4.0.7	4.0.x	4.6
4.0.6	4.0.x	4.8
4.0.6	4.0.x	4.6
4.0.5	4.0.x	4.6
4.0.4	4.0.x	4.8
4.0.4	4.0.x	4.6
4.0.2	4.0.x	4.8
4.0.2	4.0.x	4.6
4.0.0	4.0.x	4.6
1.5.0	3.5	4.6
1.5.0	3.5	3.11

1.5.0	3.0.1	4.5
1.5.0	3.0.1	3.11

Support matrix

Service architecture for IBM Cloud Pak for Data

IBM Cloud Pak for Data

Learn about the components that comprise the service on IBM Cloud Pak® for Data and how data flows through the service.

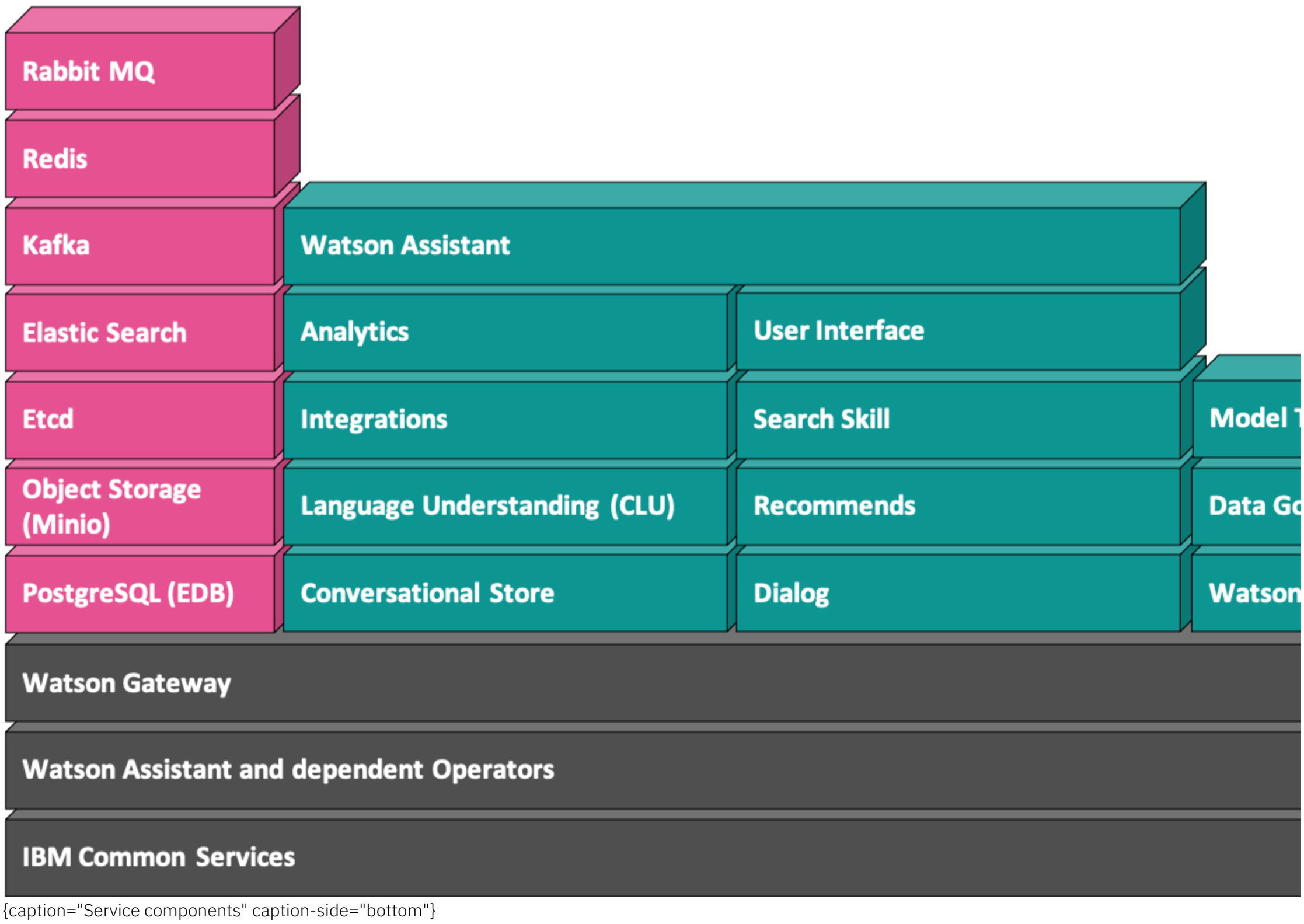
The service consists of the following types of resources:

- Microservices
- Supporting data stores
- Training component

The service uses the following patterns to communicate and pass information among its resources:

- **REST API**: Sends representational state transfer (REST) API calls over secure HTTP.
- **gRPC**: Makes method calls by using an open source remote procedure call framework, which enables the service to call a resource that is running on another system in the cluster as if it were a local object. For more information, see gRPC.
- **LiteLinks**: Uses the LiteLinks protocol that was developed by IBM. LiteLinks has a custom service discovery layer that serves as a wrapper for an underlying Apache Thrift-based remote procedure call framework.

The following diagram illustrates the components that are used by the service.



{caption="Service components" caption-side="bottom"}

The following sections provide more detail about each resource that is used by the system. The objective is to give you information that can help you to do

initial resource planning and help you to manage changes in data needs over time.

Microservices

The service consists of the following stateless microservices:

- **Analytics**: Logs conversations that take place between the assistant and your customers. These user conversation logs are available from the Analytics page in the product and from the `/logs` API endpoint. Introduced with the 1.5.0 release. The Analytics feature relies on a microservice that is supported only on Red Hat OpenShift 4.5 and later.

- **CLU**: A Conversational Language Understanding interface, also known as Natural Language Understanding (NLU), that is an entry point to the language understanding pipeline, which is where text is analyzed to find intent and entity mentions. The Store microservice uses the LiteLinks protocol to call this microservice to initiate machine learning training. The NLU microservice controls the lifecycle of the machine learning models. It maps the workspace ID that is used by the Store microservice to the machine language understanding the pipeline's internal ID. The NLU microservice uploads the training data for machine learning models to the MinIO object storage. When a language model needs to be trained, such as after intents are edited, the NLU microservice calls the Master microservice. When the updated model is available, NLU calls the TAS microservice to analyze the user input that is provided by the customer. The workspace ID has a UUID format. The internal IDs of models in the language understanding pipeline typically follow the regular expression pattern, `vn-[0-9a-f]*-[0-9a-f]*`.

- **CLU Embedding**: Provides the word embeddings for other microservices in the language understanding pipeline. The TAS, ED-MM, and training pods call the CLU Embedding microservice by using the gRPC protocol.

- **Dialog**: Processes the dialog tree that is defined in a dialog skill to generate responses to user inputs. Based on the context (variables that are passed from previous conversation turns) and the intents and entities that are recognized in the user input, this service follows the dialog tree in the skill to find an appropriate response. Dialog is also responsible for making programmatic callouts to other services or programs. This microservice, which is sometimes referred to as the Dialog runtime, is a LiteLinks server that is called by the Store microservice. It has an in-memory cache for dialog skills and uses Redis as the cache for dialog skills.

- **ED-MM**: Recognizes contextual entities in user input. This microservice is part of the language understanding pipeline. ED-MM stands for Entities Distro model mesh. The service's implementation of the ED-MM microservice is based on the model-mesh pattern. (For more information, see Model mesh.) The ED-MM microservice is called by the TAS microservice to evaluate user input only if the dialog skill that is being processed has contextual entities in its training data. Contextual entities are bound to a context. The ED-MM microservice loads the models from MinIO. At run time, the models are used to look up the entities and validate the context. The ED-MM microservice also accesses the CLU Embedding service for word embeddings.

- **Gateway**: An adaption layer that interacts with the IBM Cloud Pak for Data API and mimics the behavior of the public IBM Cloud. Its pods are named `${release-name}-addon-assistant-gw-deployment`.

  > **Note:** The ${release-name} is `watson-assistant`.

  It serves the following functions:

  - Registers the installation of the service into IBM Cloud Pak for Data.
  - For API requests, it adds authentication data that is required by the Store microservice.
  - Notifies the Store microservice when a service instance is created or deleted.
  - When a user requests instance details from the web UI, it mimics the IBM Cloud interface so the UI microservice can return the information.

  > **Note:** When an API request comes in, it is processed by the IBM Cloud Pak for Data ingress nginx server first. The nginx server is configured to call the Gateway microservice to check authorization and obtain authentication data for the request. The nginx server adds a header to the original request, and then calls the Store microservice. However, the initial steps do not produce any logs. To see the first logs from an incoming API request, check the logs in the Store microservice.

- **Integrations**: Service that supports the built-in integrations, such as web chat and preview link. Introduced with the 1.5.0 release.

- **Master**: Controls the lifecycle of underlying intent and entity models. This microservice is part of the language understanding pipeline. The NLU microservice uses LiteLinks to send a request to the Master microservice for a new model to be trained. This type of request occurs when a dialog skill is created or an existing dialog skill is updated. After a model is successfully trained, the TAS microservice is called to load the model.

- **Numeric system entities**: Manages the numeric value recognition that is used by the system entities (such as `@sys-date` or `@sys-number`). Introduced with the 1.5.0 release.

- **Recommends**: Supports recommendations from Watson, such as dictionary-based entity synonyms and intent conflicts. Used only at authoring time from the web UI. When a user requests synonym recommendations for an entity, for example, the UI microservice calls the Store microservice. The Store microservice forwards the request to the Recommends microservice. The Recommends microservice looks up the synonyms for the current

word by using the embeddings that are stored in MongoDB, and stores them in a Redis cache. This microservice sends a response with a list of synonyms to the Store microservice. A similar workflow is used to identify intent conflicts in a dialog skill.

- **Skill-search**: Manages API calls to a Discovery service instance that is enabled in the same cluster. When a v2 `/message` API call reaches the Store microservice, the Store retrieves session state information from Redis and start processing skills. When the assistant that is being processed has a search skill, the Store microservice calls this microservice over HTTPS REST. This microservice queries the Discovery instance and converts the output that is returned by Discovery to the v2 `/message` API schema.
- **Spellchecker**: Corrects spelling mistakes that are made in user input that is submitted with `/message` requests. This autocorrection feature is enabled automatically for English dialog skills, and can be turned on for French skills. This microservice provides basic spell-checking capabilities by using correction techniques such as edit-distance from vocabulary word and generic language models. If enabled, the TAS microservice calls the Spellchecker by using gRPC before it performs recognition of intents and entities in the user input. Spellchecker does not depend on data stores and it does not call any other microservice.
- **Store**: Handles all assistant API calls. This microservice either processes the request itself or calls the other microservices that are needed to process the request. For example, if a customer submits user input with a v1 `/message` API call, the request is sent to and processed by the store. For stateful v2 `/message` API calls, the store retrieves session state information from Redis first. It then calls the NLU microservice to analyze the user input and identify any intent and entity references in the input. Next, the Store calls the Dialog microservice to generate the appropriate response to return to the customer. The Store microservice stores the assistants, skills, and workspace definitions in a PostgreSQL database. The Store microservice saves the state of the session (from the v2 API) in the Redis data store.
- **TAS**: Performs model inferencing, which means it identifies the best-matching intents and entities in the user input. TAS stands for Train and Serve, but it mostly serves up the existing models. The TAS microservice loads the models from MinIO storage into memory and runs the models to find intents. The TAS microservice is called from the NLU microservice by using LiteLinks. TAS and the ED-MM microservices are based on a model-mesh pattern. (For more information, see Model mesh.) If needed, TAS calls other microservices from the language understanding pipeline. Specifically, it calls SireG for tokenization, ED-MM for contextual entities (by using gRPC), and Spellcheck to correct misspellings. For intent recognition, the TAS microservice requires word-embeddings data, which is provided by the CLU Embedding microservice.
- **TF-MM**: The tensor flow model mesh microservice manages the universal sentence encoder and auto-encoder models to improve off-topic and irrelevant intent recognition. Introduced with the 1.5.0 release.
- **UI**: The web application that virtual assistant builders use to create skills and assistants.

Data sources

The microservices use the following resources:

- **Elastic**: The elastic data store stores customer messages. These user conversation logs that can be reviewed from the Analytics page or searched from the `/logs` API endpoint. Introduced with the 1.5.0 release.
- **Etcd**: A popular distributed key-value storage solution. Ectd is used by Litelinks clients and servers (Store, Dialog, NLU, Master, TAS, ED-MM) for service discovery. It is used by microservices from the language understanding pipeline (NLU, Master, TAS, ED-MM) to store metadata. For more information, see Etcd store.
- **Kafka**: A queuing system for incoming customer messages. Introduced with the 1.5.0 release.
- **PostgreSQL**: A popular relational database. This database is used only by the Store microservice and it is the primary store for workspaces, skills, and assistants. The deployment and pod names that are related to PostgreSQL are prefixed as `${release-name}-store-postgres`. For more information, see PostgreSQL data store.
- **Multicloud Object Gateway**: Multicloud Object Gateway is an object storage service that implements the Amazon S3 API. It is used by the language understanding pipeline microservices (clu-controller, clu-serving, clu-training, tf-mm, ed-mm, and dragonfly) to store and load trained models for intent and entity classification. It is used by the authoring and runtime service (store) to store versioned data of the skills and also used as a temporary storage for asynchronous uploads for the authoring experience. In Watson Assistant, Multicloud Object Gateway is often referred to as Cloud Object Storage. For more information, see Multicloud Object Gateway.
- **Redis**: An in-memory data store, often used for caching or sharing session state. Redis is used by the Store microservice for storing current conversation state for assistants. The UI microservice stores session state in Redis. The Recommends and Dialog microservices use a Redis instance as a cache.

1.5.0: The following data source is no longer used, starting with the 1.5.0 release:

- **MongoDB**: A document-oriented database. Mongo is used in read-only mode for storing embeddings and other data that is used by the Recommends microservice for synonym recommendations.

  MongoDB is a document-based, distributed database. The MongoDB database has three pods. It runs in replicaSet mode, which means that one pod runs in the coordinator role in read/write mode, and the rest of the pods run in the secondary role and are read-only. Changes from the coordinator pod are replicated to the secondary pods. During the service installation, data is loaded into the Mongo database by a Kubernetes job. No data is

written to the Mongo database after the service is installed. Only the Recommends microservices read data from Mongo. On creation, Recommends pods check whether Mongo is running, and wait until the Mongo database is loaded with the required data.

> **Note:** The process of loading data for Recommends into the Mongo database that happens as part of the installation can take 30 minutes (or more).

- The MongoDB pod names follow the convention `${release-name}-ibm-mongodb-server-[0-9]*`. For longer release names, the name is shorten to `${release-name prefix}-[a-f0-9]{4}-336f-server-*`.
- The name of the Kubernetes job that runs during the installation of the service is `${release-name}-recommends-load-mongo`.

> **Note:** The ${release-name} is `watson-assistant`.

4.7.0: The following data source is no longer used, starting with the 4.7.0 release:

- **MinIO**: MinIO is an object storage service that implements the Amazon S3 API. It is used by the language understanding pipeline microservices (NLU, Master, TAS, ED-MM, and the training pods) to store and load trained models for intent and entity classification. Data is stored in the `nlclassifier-icp` bucket. In Watson Assistant, MinIO is often referred to as `COS`, which stands for Cloud Object Storage. For more information, see MinIO.

Architecture changes

- **1.5.0**: The following changes to the architecture occurred with this release:
  - The *Analytics*, *Integrations*, and *Numeric system entities* microservices were introduced.
  - The *Elastic search* and *Kafka* data sources were introduced. The *MongoDB* data source was removed.
- **4.7.0**: The following changes to the architecture occurred with this release:
  - The *Multicloud Object Gateway* was introduced. The *MinIO* data source and *SIREG* microservice were removed.

Training component

Training a new model is a one-time, short-running, resource-heavy activity. Models are trained in pods that are created on demand. The same pods are removed after the training is completed. Because of its transient nature, the training component is not considered to be one of the standard microservices. The component, the training pods, get used only when a model needs to be trained. Training starts each time the intent user examples are added or changed in a dialog skill.

The CLU microservice notifies the Master microservice when a new machine learning model needs to be trained. The Master microservice dynamically creates training pods. It handles the training pod removal and retraining if a training run fails. It is also responsible for removing a model from MinIO if the model is deleted. The system determines whether a training run completed successfully based on a flag that is stored in MinIO storage. The Master microservice must have API access to the Docker registry where the training images are stored. The image metadata that is obtained from the registry is used to correctly start the training pods.

The training pods that are started by this component are sometimes referred to as SLAD pods. SLAD stands for Statistical Learning and Discovery, which is the name of the research group that developed the language classifier that is used by the pods. The training pod names start with `tr[12]?`, followed by the internal `vn-...` ID. The training images are named `nlclassifier-training` or `clu-training`, depending on the chart version. The `${release-name}-master-config` configuration map contains the JSON template for the training pods.

> **Note:** The ${release-name} is `watson-assistant`.

Data store details

The following sections provide more detail about how the data stores are used by the service. The objective is to help you troubleshoot issues that might arise during installation or after the service is deployed and running in a data center.

Etcd store

The service uses Etcd as a key-value store. Etcd is used by LiteLinks for service discovery and by the language understanding pipeline as configuration storage.

Etcd consists of five pods. The Etcd pod names follow the convention `${release-name}-etcd3-[0-9]*`. The service chart requires Etcd version 3 with enabled authorization.

LiteLinks service discovery

The Dialog, NLU, Master, TAS, and ED-MM microservices act as LiteLinks servers. Each LiteLinks server is registered in Etcd with its own key under the `/bluegoat/litelinks/` path. Each pod stores its metadata, such as its IP address and port, into Etcd. For example, the Dialog microservice pod with IP address 10.131.2.25 might register itself under a key named `/bluegoat/litelinks/voyager-dialog-slot-${release-name}/10.128.0.231_8089_16ea2cde43f`.

Note: The ${release-name} is `watson-assistant`.

The Store, NLU, Master, TAS, and ED-MM microservices are LiteLinks clients. (Some microservices function as both a server and client.) Each LiteLinks client reads these Etcd keys and communicates directly with the registered pod IP address and port of the server. In effect, LiteLinks clients do not use the Kubernetes DNS. As a result, Kubernetes service objects are not used for LiteLinks servers. Without the Kubernetes service objects, the readiness probes are ignored even though they are good indicators of pod health.

The following table lists the key names for each LiteLinks server.

Microservice	LiteLinks server key name
Dialog	`voyager-dialog-slot-${release-name}`
NLU	`voyager-nlu-slot-${release-name}`
Master	`voyager-master-slot-${release-name}`
TAS	`tas-runtime-slot-${release-name}`
ED-MM	`${release-name}-ed-mm`

LiteLinks server Etcd paths

Microservice pods that are either LiteLinks clients or servers contain an initContainer that, on pod creation, checks, and waits until Etcd is operational and contains configuration entries.

Configuration storage

The microservices in the language understanding pipeline use Etcd to store some configuration values.

Each microservice has its own path in Etcd. Other metadata about models, such as the instance in which a model is loaded, is stored under different keys under the Etcd path.

Microservice	Etcd path
NLU	`/bluegoat/voyager-nlu/voyager-nlu-slot-${release-name}`
Master	`/bluegoat/bluegoat-master/voyager-master-slot-${release-name}`
TAS	`/bluegoat/tas-runtime/tas-runtime-slot-${release-name}`
ED-MM	`/bluegoat/tas-runtime/${release-name}-ed-mm/`

Microservice configuration store Etcd paths

The configuration values per microservice are stored under a `/config` subpath.

MinIO

MinIO is an enterprise-grade object storage service that implements the Amazon S3 API. The service chart runs MinIO in distributed mode with four pods. With this configuration, MinIO functions fully in read/write mode if more than half, meaning three or more, of the pods are available. Its function is degraded to read-only mode if only half (two) of the pods are available.

MinIO pods are named `${release-name}-clu-minio-[0-9]*`. CLU, which stands for Conversational Language Understanding, is included in the pod names to associate the MinIO pods with the language understanding pipeline. It is only the pipeline microservices, such as NLU, Master, TAS, ED-MM, and the training pods that use MinIO.

PostgreSQL data stores

The Postgres data store is based on *stolon*, which is a cloud-native PostgreSQL manager for PostgreSQL high availability. (For more information, see the

GitHub sorintlab repo.) The Postgres store consists of the following kinds of pods:

- keeper: These pods run the PostgreSQL database. There are three keeper pods. One of the three is selected as the coordinator keeper. The coordinator keeper handles all of the SQL queries. The remaining pods are on standby and their state is updated by the coordinator keeper pod.

  The keeper pod names follow the convention of `${release-name}-store-postgres-keeper-*`. If the release name is long, the pod name might be shortened to something like `${release-name prefix}-[a-f0-9]{4}-st-a617-keeper-*`.

  > **Note:** The ${release-name} is `watson-assistant`. The shortened name is `watson-ass`.

- proxy: These pods are the entry points that are used by the Store microservice. The proxy pods route traffic to the coordinator keeper pod.
- sentinel: These pods are the ones that decide which of the keepers is the coordinator.

To manage the PostgreSQL cluster, use `stolonctl` commands inside the keeper pods. The metadata about the PostgreSQL configuration is stored in the configmap named `stolon-cluster-${release-name}`.

Postgres is used by the Store microservice to store assistants, skills, and workspaces. If PostgreSQL and the Store microservice are running, even if nothing else is working, you can export your skills from the product and save them.

During installation, the Postgres database is created. The Store microstore user is also created. You can specify the name of the database, the name of the user, and a corresponding password if you want by overriding configuration settings in the `values.yaml` configuration file.

The following table lists the configuration settings that are used by the Store microservice to connect to the PostgresSQL and for PostgreSQL initialization at installation time.

Configuration setting name	Description	Default value
global.postgres.store.auth.user	Username that is used by the Store microservice	`store_icp_${release-name}`
global.postgres.store.auth.authSecretName	Name of the Kubernetes secret with password for the Store user. The default value is null because a password is randomly generated.	null
global.postgres.store.database	Name of the database that is used by the Store microservice	`conversation_icp_${release-name}`

Postrgres datasource configuration settings

Model mesh

The TAS and ED-MM microservices use a model-mesh pattern. In model mesh, each pod contains a set of loaded NLP models. Managing where models are loaded is handled by a model-mesh library. During normal operations, the model is loaded in one pod only. If traffic to the model cannot be handled by a single pod, the model is loaded into other pods also. The metadata about where the models are loaded is stored in Etcd.

TAS and ED-MM belong to separate model-mesh groups. Each group has an independent set of pods and loaded models.

Here's how model mesh functions in the ED-MM microservice pod group. For the ED-MM microservice, you might have two pods, A and B. A gRPS request for help with contextual entity recognition is received through Kubernetes DNS. The request goes to pod A, but the contextual entity model that is needed is loaded in pod B. The model mesh on pod A forwards the request to pod B by using LiteLinks. Pod B uses the appropriate model to identify contextual entities in the input, and then sends a response to pod A. Pod A sends the information in a response back to the service that sent the initial gRPC request.

Model mesh functions in the same way for the TAS microservice. The difference is that incoming requests to TAS are sent by using LiteLinks and not gRPC. As a result, the NLU and Master microservices have no way of knowing which models are loaded in which TAS pods.

Backing up and restoring data for IBM On-premises

IBM Cloud Pak for Data | IBM Software Hub

You can back up and restore the data that is associated with your installation in IBM On-premises.

The following table lists the upgrade paths that are supported by the scripts.

Version in use	Version that you can upgrade to
5.0.x	5.1.x

4.8.x	5.0.x or 5.1.x
4.7.x	4.8.x or 5.0.x
4.6.x	4.7.x or 4.8.x
4.5.x	4.6.x or 4.7.x
4.0.x	4.5.x or 4.6.x

Upgrade paths supported by scripts

Simpler ways to complete the upgrade is described in the following topics:

- Upgrading watsonx Assistant to Version 5.1.x
- Upgrading watsonx Assistant to Version 5.0.x
- Upgrading watsonx Assistant to Version 4.8.x
- Upgrading watsonx Assistant to Version 4.7.x
- Upgrading watsonx Assistant to Version 4.6.x

⚠ **Important:** If you are upgrading from 4.6.4 or earlier to the latest version, you must upgrade to 4.6.5 before you upgrade to the latest release.

The primary data storage is a PostgreSQL database.

Choose one of the following ways to manage the back up of data:

- Kubernetes CronJob: Use the `$INSTANCE-store-cronjob` cron job that is provided for you.
- backupPG.sh script: Use the `backupPG.sh` bash script.
- pg_dump tool: Run the `pg_dump` tool on each cluster directly. This is a manual option that gives you control over the process.

Note: When you back up data with one of these procedures before you upgrade from one version to another, the workspace IDs of your skills are preserved, but the service instance IDs and credentials change.

Before you begin

- When you create a backup by using this procedure, the backup includes all of the assistants and skills from all of the service instances. It can include skills and assistants to which you do not have access.
- The access permissions information of the original service instances is not stored in the backup. Meaning original access rights, which determine who can see a service instance and who cannot, are not preserved.
- You cannot use this procedure to back up the data that is returned by the search integration. Data that is retrieved by the search integration comes from a data collection in a Discovery instance. See the Discovery documentation to find out how to back up its data.
- If you back up and restore or otherwise change the Discovery service that your search integration connects to, then you cannot restore the search integration, but must re-create it. When you set up a search integration, you map sections of the assistant's response to fields in a data collection that is hosted by an instance of Discovery on the same cluster. If the Discovery instance changes, your mapping to it is broken. If your Discovery service does not change, then the search integration can continue to connect to the data collection.
- The tool that restores the data clears the current database before it restores the backup. Therefore, if you might need to revert to the current database, create a backup of it first.
- The target IBM Cloud Pak for Data cluster where you restore the data must have the same number of provisioned service instances as the environment from which you back up the database. To verify in the IBM Cloud Pak for Data web client, select **Services** from the main navigation menu, select **Instances**, and then open the **Provisioned instances** tab. If more than one user created instances, then ask the other users who created instances to log in and check the number that they created. You can then add up the total sum of instances for your deployment. Not even an administrative user can see instances that were created by others from the web client user interface.

Backing up data by using the CronJob

A CronJob named `$INSTANCE-store-cronjob` is created and enabled for you automatically when you deploy the service. A CronJob is a type of Kubernetes controller. A CronJob creates Jobs on a repeating schedule. For more information, see CronJob in the Kubernetes documentation.

The store CronJob creates the `$INSTANCE-backup-job-$TIMESTAMP` jobs. Each `$INSTANCE-backup-job-$TIMESTAMP` job deletes old logs and runs a backup of the store PostgreSQL database. PostgreSQL provides a `pg_dump` tool that creates a backup. To create a backup, the `pg_dump` tool sends the database

contents to `stdout`, which you can then write to a file. The `pg_dump` tool creates the backups with the `pg_dump` command and stores them in a persistent volume claim (PVC) named `$INSTANCE-store-db-backup-pvc`.

> **Note:** You are responsible for moving the backup to a more secure location after its initial creation, preferably a location that can be accessed outside of the cluster where the backups cannot be deleted easily. Ensure this happens for all environments, especially for Production clusters.

The following table lists the configuration values that control the backup cron job. You can edit these settings by editing the cron job after the service is deployed by using the `oc edit cronjob $INSTANCE-store-cronjob` command.

Variable	Description	Default value
store.backup.suspend	If True, the cron job does not create any backup jobs.	`False`
store.backup.schedule	Specifies the time of day at which to run the backup jobs. Specify the schedule by using a cron expression. For example, `{minute} {hour} {day} {month} {day-of-week}` where `{day-of-week}` is specified as `0`=Sunday, `1`=Monday, and so on. The default schedule is to run every day at 11 PM.	`0 23 * * *`
store.backup.history.jobs.success	The number of successful jobs to keep.	`30`
store.backup.history.jobs.failed	The number of failed jobs to keep in the job logs.	`10`
store.backup.history.files.weekly_backup_day	A day of the week is designated as the weekly backup day. 0=Sunday, 1=Monday, and so on.	`0`
store.backup.history.files.keep_weekly	The number of backups to keep that were taken on weekly_backup_day.	`4`
store.backup.history.files.keep_daily	The number of backups to keep that were taken on all the other days of the week.	`6`

Cron job variables

Accessing backed-up files from Portworx

To access the backup files from Portworx, complete the following steps:

1. Get the name of the persistent volume that is used for the PostgreSQL backup:

   ```
   oc get pv |grep $INSTANCE-store
   ```

   This command returns the name of the persistent volume claim where the store backup is located, such as `pvc-d2b7aa93-3602-4617-acea-e05baba94de3`. The name is referred to later in this procedure as the `$pv_name`.

2. Find nodes where Portworx is running:

   ```
   oc get pods -n kube-system -o wide -l name=portworx-api
   ```

3. Log in as the core user to one of the nodes where Portworx is running:

   ```
   ssh core@<node hostname>
   sudo su -
   ```

4. Make sure that the persistent volume is in a detached state and that no store backups are scheduled to occur during the time you plan to transfer the backup files.

   Remember, backups occur daily at 11 PM (in the time zone that is configured for the nodes) unless you change the schedule by editing the value of the `postgres.backup.schedule` configuration parameter. You can run the `oc get cronjobs` command to check the current schedule for the `$RELEASE-backup-cronjob` job. In the following command, `$pvc_node` is the name of the node that you discovered in the first step of this task:

   ```
   pxctl volume inspect $pv_name |head -40
   ```

5. Attach the persistent volume to the host:

```
pxctl host attach $pv_name
```

6. Create a folder where you want to mount the node:

```
mkdir /var/lib/osd/mounts/voldir
```

7. Mount the node:

```
pxctl host mount $pv_name --path /var/lib/osd/mounts/voldir
```

8. Change the directory to `/var/lib/osd/mounts/voldir`. Transfer backup files to a secure location. Afterward, exit the directory. Unmount the volume:

```
pxctl host unmount --path /var/lib/osd/mounts/voldir $pv_name
```

9. Detach the volume from the host:

```
pxctl host detach $pv_name
```

10. Make sure that the volume is in the detached state. Otherwise, subsequent backups fail:

```
pxctl volume inspect $pv_name |head -40
```

Accessing backed-up files from Red Hat OpenShift Container Storage

To access the backup files from Red Hat OpenShift Container Storage (OCS), complete the following steps:

1. Create a volume snapshot of the persistent volume claim that is used for the PostgreSQL backup:

```
cat <<EOF | oc apply -f -
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: wa-backup-snapshot
spec:
  source:


    persistentVolumeClaimName: ${INSTANCE_NAME}-store-db-backup-pvc


  volumeSnapshotClassName: ocs-storagecluster-rbdplugin-snapclass
EOF
```

2. Create a persistent volume claim from the volume snapshot:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wa-backup-snapshot-pvc
spec:
  storageClassName: ocs-storagecluster-ceph-rbd
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: wa-backup-snapshot
  resources:
    requests:
      storage: 1Gi
EOF
```

3. Create a pod to access the persistent volume claim:

```
cat <<EOF | oc apply -f -
kind: Pod
apiVersion: v1
metadata:
  name: wa-retrieve-backup
spec:
  volumes:
    - name: backup-snapshot-pvc
      persistentVolumeClaim:
       claimName: wa-backup-snapshot-pvc
  containers:
    - name: retrieve-backup-container
      image: cp.icr.io/cp/watson-assistant/conan-tools:20210630-0901-signed@sha256:e6bee20736bd88116f8dac96d3417afdfad477af21702217f8e6321a99190278
      command: ['sh', '-c', 'echo The pod is running && sleep 360000']
      volumeMounts:
        - mountPath: "/watson_data"
          name: backup-snapshot-pvc
EOF
```

4. If you do not know the name of the backup file that you want to extract and are unable to check the most recent backup cron job, run the following command:

   ```
   oc exec -it wa-retrieve-backup -- ls /watson_data
   ```

5. Transfer the backup files to a secure location:

   ```
   kubectl cp wa-retrieve-backup:/watson_data/${FILENAME} ${SECURE_LOCAL_DIRECTORY}/${FILENAME}
   ```

6. Run the following commands to clean up the resources that you created to retrieve the files:

   ```
   oc delete pod wa-retrieve-backup
   oc delete pvc wa-backup-snapshot-pvc
   oc delete volumesnapshot wa-backup-snapshot
   ```

Extracting PostgreSQL backup by using a debug pod

To extract PostgreSQL backup using a debug pod, complete the following steps:

1. Get the name of the store cronjob pod:

   ```
   export STORE_CRONJOB_POD=`oc get pods -l component=store-cronjob --no-headers | awk 'NR==1{print $1}'`
   ```

2. View the list of available store backups to identify the most recent backup:

   ```
   oc debug ${STORE_CRONJOB_POD}
   ls /store-backups/
   ```

   In the list of store backups, you can find the latest backup with the help of the timestamps.

3. While the debug pod listed in Step 2 remains active, in a separate terminal session, set the `STORE_CRONJOB_POD` variable to match the name of the store cronjob pod returned in Step 1:

   ```
   export STORE_CRONJOB_POD=`oc get pods -l component=store-cronjob --no-headers | awk 'NR==1{print $1}'`
   ```

4. Export and save the `STORE_DUMP_FILE` variable to the name of the most recent `store.dump_YYYYMMDD-TIME` file from `Step 2`:

   ```
   export STORE_DUMP_FILE=store.dump_YYYYMMDD-TIME
   ```

5. Copy the `store.dump_YYYYMMDD-TIME` file to a directory in a secure location on your system:

   ```
   `oc cp ${STORE_CRONJOB_POD}-debug:/store-backups/${STORE_DUMP_FILE} ${STORE_DUMP_FILE}`
   ```

   You must verify that you copied the `store.dump_YYYYMMDD-TIME` file to the right directory by running the `ls` command.

Backing up data by using the script

You cannot backup data by using script in watsonx Assistant for IBM Cloud Pak® for Data 4.6.3 or later.{ .note}

The `backupPG.sh` script gathers the pod name and credentials for one of your PostgreSQL pods. Then, the `backupPG.sh` script uses the PostgreSQL pod to run the `pg_dump` command.

To back up data by using the provided script, complete the following steps:

1. Download the `backupPG.sh` script.

   Go to GitHub, and find the directory for your version to find the file.

   > **Note:** If the `backupPG.sh` script doesn't exist in the directory of your version, backup your data by using KubernetesCronJob or pg_dump tool.

2. Log in to the Red Hat OpenShift project namespace where you installed the product.

3. Run the script:

   ```
   ./backupPG.sh --instance ${INSTANCE} > ${BACKUP_DIR}
   ```

   Replace the following values in the command:

   - `${BACKUP_DIR}`: Specify a file where you want to write the downloaded data. Be sure to specify a backup directory in which to store the file. For example, `/bu/backup-file-name.dump` creates a backup directory named `bu`.
   - `--instance ${INSTANCE}`: Select the specific instance to be backed up.

If you prefer to back up data by using the PostgreSQL tool directly, you can complete the procedure to back up data manually.

Backing up data manually

Complete the steps in this procedure to back up your data by using the PostgreSQL tool directly.

To back up your data, complete these steps:

1. Fetch a running PostgreSQL pod:

   Only for version 4.8.8, 5.1.0, and all future versions:

   ```
   oc get pods -l app=${INSTANCE}-postgres-16 -o jsonpath="{.items[0].metadata.name}"
   ```

   For other versions, use:

   ```
   oc get pods -l app=${INSTANCE}-postgres -o jsonpath="{.items[0].metadata.name}"
   ```

   Replace ${INSTANCE} with the instance of the deployment that you want to back up.

2. Perform the following two steps only if you have **version 5.0.0 or 4.8.5 and before** :

   a. Fetch the store VCAP secret name:

   ```
   oc get secrets -l component=store,app.kubernetes.io/instance=${INSTANCE} -o=custom-columns=NAME:.metadata.name | grep store-vcap
   ```

   b. Fetch the PostgreSQL connection values. You will pass these values to the command that you run in the next step. You must have `jq` installed.

   - To get the database:

     ```
     oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.database'
     ```

   - To get the hostname:

     ```
     oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.host'
     ```

   - To get the username:

     ```
     oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.username'
     ```

- To get the password:

```
oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.password'
```

3. Perform the following two steps only if you have **version 4.8.6 or 5.0.1 and later** :

   a. Fetch the store connection secret name:

```
oc get secrets -l component=store-subsystem,app.kubernetes.io/instance=${INSTANCE} -o=custom-columns=NAME:.metadata.name | grep store-datastore-connection
```

   b. Fetch the PostgreSQL connection values. You will pass these values to the command that you run in the next step. You must have `jq` installed.

   - To get the database:

```
oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.store_vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.database'
```

   - To get the hostname:

```
oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.store_vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.host'
```

   - To get the username:

```
oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.store_vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.username'
```

   - To get the password:

```
oc get secret $VCAP_SECRET_NAME -o jsonpath="{.data.store_vcap_services}" | base64 --decode | jq --raw-output '.["user-provided"][]|.credentials|.password'
```

4. Run the following command:

```
oc exec $KEEPER_POD -- bash -c "export PGPASSWORD='$PASSWORD' && pg_dump -Fc -h $HOSTNAME -d $DATABASE -U $USERNAME" > ${BACKUP_DIR}
```

   The following lists describe the arguments. You retrieved the values for some of these parameters in the previous step:

   Only for version 4.8.8, 5.1.0, and all future versions:

   Use `$KEEPER_POD` : Any PostgreSQL 16 pod in your instance.

   For other versions:

   Use `$KEEPER_POD` : Any PostgreSQL pod in your instance.

   For all versions:

   - `${BACKUP_DIR}` : Specify a file where you want to write the downloaded data. Be sure to specify a backup directory in which to store the file. For example, `/bu/backup-file-name.dump` creates a backup directory named `bu` .
   - `$DATABASE` : The store database name that was retrieved from the Store VCAP secret in step 3.
   - `$HOSTNAME` : The hostname that was retrieved from the Store VCAP secret in step 3.
   - `$USERNAME` : The username that was retrieved from the Store VCAP secret in step 3.
   - `$PASSWORD` : The password that was retrieved from the Store VCAP secret in step 3.

   To see more information about the `pg_dump` command, you can run this command:

```
oc exec -it ${KEEPER_POD} -- pg_dump --help
```

5. Take a backup of the secret that contains the encryption key. Ignore this step if the below mentioned secret is not available in that release.

```
oc get secret -l service=conversation,app=$INSTANCE-auth-encryption
oc get secret $INSTANCE-auth-encryption -o yaml > auth-encryption-secret.yaml
```

Restoring data

IBM created a restore tool called `pgmig` . The tool restores your database backup by adding it to a database that you choose. It also upgrades the schema to the one that is associated with the version of the product where you restore the data. Before the tool adds the backed-up data, it removes the data for all

instances in the current service deployment, so any spares are also removed.

Prerequisite:

Setup `auth-encryption-secret` that you backed up earlier.

```
oc apply -f auth-encryption-secret.yaml
oc get secret -l service=conversation,app=$INSTANCE-auth-encryption
```

1. Install the target IBM Cloud Pak for Data cluster to which you want to restore the data.

   From the web client for the target cluster, create one service instance of for each service instance that was backed up on the old cluster. The target IBM Cloud Pak for Data cluster must have the same number of instances as there were in the environment where you backed up the database.

2. Back up the current database before you replace it with the backed-up database.

   The tool clears the current database before it restores the backup. So, if you might need to revert to the current database, be sure to create a backup of it first.

3. Go to the backup directory that you specified in the `${BACKUP_DIR}` parameter in the previous procedure.

4. Run the following command to download the `pgmig` tool from the GitHub Watson Developer Cloud Community repository.

   ⚠ **Important:** In the first command, update `<WA_VERSION>` to the version that you want to restore. For example, update `<WA_VERSION>` to `4.6.0` if you want to restore 4.6.0.

   ```
   wget https://github.com/watson-developer-cloud/community/raw/master/watson-assistant/data/<WA_VERSION>/pgmig
   chmod 755 pgmig
   ```

5. Create the following two configuration files and store them in the same backup directory:

   - `resourceController.yaml`: The Resource Controller file keeps a list of all provisioned instances. See Creating the resourceController.yaml file.
   - `postgres.yaml`: The PostgreSQL file lists details for the target PostgreSQL pods. See Creating the postgres.yaml file.

6. Get the secret:

   Only for version 4.8.8, 5.1.0, and all future versions:

   ```
   oc get secret ${INSTANCE}-postgres-16-ca -o jsonpath='{.data.ca\.crt}' | base64 -d | tee ${BACKUP_DIR}/ca.crt | openssl x509 -noout -text
   ```

   For other versions:

   ```
   oc get secret ${INSTANCE}-postgres-ca -o jsonpath='{.data.ca\.crt}' | base64 -d | tee ${BACKUP_DIR}/ca.crt | openssl x509 -noout -text
   ```

   - Replace `${INSTANCE}` with the name of the instance that you want to back up.
   - Replace `${BACKUP_DIR}` with the directory where the `postgres.yaml` and `resourceController.yaml` files are located.

7. Copy the files that you downloaded and created in the previous steps to any existing directory on a PostgreSQL pod.

   a. **Only for version 4.8.8, 5.1.0, and all future versions:**

   Run the following command to find PostgreSQL pods:

   ```
   oc get pods | grep ${INSTANCE}-postgres-16
   ```

   b. **For other versions:**

   Run the following command to find PostgreSQL pods:

   ```
   oc get pods | grep ${INSTANCE}-postgres
   ```

   c. The files that you must copy are `pgmig`, `postgres.yaml`, `resourceController.yaml`, `ca.crt` (the secret file that is generated in step 6), and the file that you created for your downloaded data. Run the following commands to copy the files.

   Note: If you are restoring data to a stand-alone IBM Cloud Pak for Data cluster, then replace all references to `oc` with `kubectl` in these

sample commands.

```
oc exec -it ${POSTGRES_POD} -- mkdir /controller/tmp
oc exec -it ${POSTGRES_POD} -- mkdir /controller/tmp/bu
oc rsync ${BACKUP_DIR}/ ${POSTGRES_POD}:/controller/tmp/bu/
```

- Replace `${POSTGRES_POD}` with the name of one of the PostgreSQL pods from the previous step.

8. Stop the store deployment by scaling the store deployment down to 0 replicas:

```
oc scale deploy ibm-watson-assistant-operator -n ${OPERATOR_NS} --replicas=0
oc get deployments -l component=store
```

Make a note of how many replicas there are in the store deployment:

```
oc scale deployment ${STORE_DEPLOYMENT} --replicas=0
```

9. Initiate the execution of a remote command in the PostgreSQL pod:

```
oc exec -it ${POSTGRES_POD} /bin/bash
```

10. Run the `pgmig` tool:

Only for version 4.8.8, 5.1.0, and all future versions:

```
cd /controller/tmp/bu
export PG_CA_FILE=/controller/tmp/bu/ca.crt
./pgmig --resourceController resourceController.yaml --target postgres.yaml --source <backup-file-name.dump>
export ENABLE_ICP=true
```

For other versions:

```
cd /controller/tmp/bu
export PG_CA_FILE=/controller/tmp/bu/ca.crt
./pgmig --resourceController resourceController.yaml --target postgres.yaml --source <backup-file-name.dump>
```

- Replace `<backup-file-name.dump>` with the name of the file that you created for your downloaded data.

For more command options, see PostgreSQL migration tool details.

As the script runs, you are prompted for information that includes the instance on the target cluster to which to add the backed-up data. The data on the instance you specify is removed and replaced. If there are multiple instances in the backup, you are prompted multiple times to specify the target instance information.

11. Scale the store deployment back up:

```
oc scale deployment ${STORE_DEPLOYMENT} --replicas=${ORIGINAL_NUMBER_OF_REPLICAS}
oc scale deploy ibm-watson-assistant-operator -n ${OPERATOR_NS} --replicas=1
```

You might need to wait a few minutes before the data your restored is visible from the web interface.

12. After you restore the data, you must train the backend model. For more information about retraining your backend model, see Retraining your backend model.

Creating the resourceController.yaml file

The **resourceController.yaml** file contains details about the new environment where you are adding the backed-up data. Add the following information to the file:

```
accessTokens:
  - value
  - value2
host: localhost
port: 5000
```

To add the values that are required but currently missing from the file, complete the following steps:

1. To get the accessTokens values list, you need to get a list of bearer tokens for the service instances.

   - Log in to the IBM Cloud Pak for Data web client.
   - From the main IBM Cloud Pak for Data web client navigation menu, select **My instances**.
   - On the **Provisioned instances** tab, click your instance.
   - In the Access information of the instance, find the **Bearer token**. Copy the token and paste it into the accessTokens list.

   A bearer token for an instance can access all instances that are owned by the user. Therefore, if a single user owns all of the instances, then only one bearer token is required.

   If the service has multiple instances, each owned by a different user, then you must gather bearer tokens for each user who owns an instance. You can list multiple bearer token values in the `accessTokens` section.

2. To get the host information, you need details for the pod that hosts the UI component:

   ```
   oc describe pod -l component=ui
   ```

   Look for the section that says `RESOURCE_CONTROLLER_URL: https://${release-name}-addon-assistant-gateway-svc.zen:5000/api/ibmcloud/resource-controller`.
   For example, you can use a command like this to find it:

   ```
   oc describe pod -l component=ui | grep RESOURCE_CONTROLLER_URL
   ```

   Copy the host that is specified in the `RESOURCE_CONTROLLER_URL`. The host value is the `RESOURCE_CONTROLLER_URL` value, excluding the protocol at the beginning and everything from the port to the end of the value. For example, for the previous example, the host is `${release-name}-addon-assistant-gateway-svc.zen`.

3. To get the port information, again check the RESOURCE_CONTROLLER_URL entry. The port is specified after `<host>:` in the URL. In this sample URL, the port is `5000`.

4. Paste the values that you discovered into the YAML file and save it.

Creating the postgres.yaml file

The **postgres.yaml** file contains details about the PostgreSQL pods in your target environment (the environment where you restore the data). Add the following information to the file:

```
host: localhost
port: 5432
database: store
username: user
su_username: admin
su_password: password
```

To add the values that are required but currently missing from the file, complete the following steps:

1. **For version 4.8.6 or 5.0.1 and later** :

   To get information about the `host`, you must get the Store datastore connection strings secret.

   ```
   oc get secret ${INSTANCE}-store-datastore-connection-strings -o jsonpath='{.data.store_vcap_services}' | base64 -d
   ```

   For version 5.0.0 or 4.8.5 and before :

   To get information about the `host`, you must get the Store VCAP secret.

   ```
   oc get secret ${INSTANCE}-store-vcap -o jsonpath='{.data.vcap_services}' | base64 -d
   ```

   The `get` command returns information about the Redis and PostgreSQL databases. Look for the segment of JSON code for the PostgreSQL database, named `pgservice`. It looks like this:

   ```
   {
     "user-provided":[
       {
   ```

```
    "name": "pgservice",
    "label": "user-provided",
    "credentials":
    {
      "host": "${INSTANCE}-rw",
      "port": 5432,
      "database": "conversation_pprd_${INSTANCE}",
      "username": "${dbadmin}",
      "password": "${password}"
    }
  }
 ],
}
```

2. Copy the values for user-provided credentials (`host` , `port` , `database` , `username` , and `password`).

   You can specify the same values that were returned for `username` and `password` as the `su_username` and `su_password` values.

   The updated file looks something like this:

   Only for version 4.8.8, 5.1.0, and all future versions:

   ```
   host: wa_inst-postgres-16-rw
   port: 5432
   database: conversation_pprd_wa_inst
   username: dbadmin
   su_username: dbadmin
   su_password: mypassword
   ```

   For other versions:

   ```
   host: wa_inst-postgres-rw
   port: 5432
   database: conversation_pprd_wa_inst
   username: dbadmin
   su_username: dbadmin
   su_password: mypassword
   ```

3. Save the `postgres.yaml` file.

PostgreSQL migration tool details

The following table lists the arguments that are supported by the `pgmig` tool:

Argument	Description
-h, --help	Command usage
-f, --force	Erase data if present in the target Store
-s, --source string	Backup file name
-r, --resourceController string	Resource Controller configuration file name
-t, --target string	Target PostgreSQL server configuration file name
-m, --mapping string	Service instance-mapping configuration file name (optional)
--testRCConnection	Test the connection for Resource Controller, then exit
--testPGConnection	Test the connection for PostgreSQL server, then exit
-v, --version	Get Build version

pgmig tool arguments

The mapping configuration file

After you run the script and specify the mappings when prompted, the tool generates a file that is named `enteredMapping.yaml` in the current directory. This file reflects the mapping of the old cluster details to the new cluster based on the interactive inputs that were provided while the script was running.

For example, the YAML file contains values like this:

```
instance-mappings:
  00000000-0000-0000-0000-001570184978: 00000000-0000-0000-0000-001570194490
```

Where the first value (`00000000-0000-0000-0000-001570184978`) is the instance ID in the database backup and the second value (`00000000-0000-0000-0000-001570194490`) is the ID of a provisioned instance in the service on the system.

You can pass this file to the script for subsequent runs of the script in the same environment. Or you can edit it for use in other back up and restore operations. The mapping file is optional. If it is not provided, the tool prompts you for the mapping details based on information you provide in the YAML files.

Retraining your backend model

Per the number of models in your assistant, you can use one of the following options to retrain your backend model:

- Retrain your backend model manually
- Auto-retrain your backend model

Retrain your backend model manually

When you open a dialog skill after a change in the training data, training is initiated automatically. Give the skill time to retrain on the restored data. It usually takes less than 10 minutes to get trained. The process of training a machine learning model requires at least one node to have 4 CPUs that can be dedicated to training. Therefore, open restored assistants and skills during low traffic periods and open them one at a time. If the assistant or dialog skill does not respond, then modify the workspace (for example, add an intent and then remove it). Check and confirm.

Auto-retrain your backend model

When you have a large number of models to retrain, you can use the auto-retrain-all job to train the backend model. To learn more about the auto-retrain-all job and its implementation, refer to the following topics:

- Before you begin
- Planning
- Procedure
- Speeding up the retrain process

Before you begin

Before you begin the auto-retrain-all job, you must ensure that the PostgreSQL database and Cloud Object Storage (Cloud Object Storage), which stores your action and dialog skills along with their snapshots, are active and not corrupted. In addition, you must ensure that your assistants do not receive or send any data during the auto-retrain-all job.

Planning

To get a good estimation of the duration that is required to complete the auto-retrain-all job, you can use the `calculate_autoretrain_all_job_duration.sh` script:

Only for version 5.1.0, 5.0.3, 4.8.8, and all future versions:

Specify the namespace where assistant is installed in the `PROJECT_CPD_INST_OPERANDS` key in the script below.

```
#!/bin/bash

calculate_duration() {
 local input_variable="$1"
 DURATION=$(("$NUM_OF_WORKSPACES_TO_TRAIN"*60 / (input_variable * 2) + "$NUM_OF_WORKSPACES_TO_TRAIN" * 2))
}

export PROJECT_CPD_INST_OPERANDS=<namespace where Assistant is installed>

ETCD_ENDPOINTS=$(oc get secret wa-cluruntime-datastore-connection-strings -n ${PROJECT_CPD_INST_OPERANDS} -o jsonpath="{.data.etcd}" | base64 --decode | jq -r
'.endpoints')
```

```
NUM_OF_WORKSPACES_TO_TRAIN=$(oc exec wa-etcd-0 -n ${PROJECT_CPD_INST_OPERANDS} -- bash -c "
 password=\"\$( cat /var/run/credentials/pass.key)\"
 etcdctl_user=\"root:\$password\"
 export ETCDCTL_USER=\"\$etcdctl_user\"

 ETCDCTL_API=3 etcdctl --cert=/etc/etcdtls/operator/etcd-tls/etcd-client.crt --key=/etc/etcdtls/operator/etcd-tls/etcd-client.key --cacert=/etc/etcdtls/operator/etcd-tls/etcd-client-ca.crt --endpoints=${ETCD_ENDPOINTS} get  --prefix  /bluegoat/voyager-nlu/voyager-nlu-slot-wa/workspaces/ --keys-only | sed '/^$/d' | wc -l")

echo "Number of workspaces to train $NUM_OF_WORKSPACES_TO_TRAIN"

calculate_duration 5
DURATION_5=$DURATION

calculate_duration 10
DURATION_10=$DURATION

calculate_duration 15
DURATION_15=$DURATION

echo "Approximate duration of the auto retrain all job if you have 5 Training pods: $DURATION_5 seconds"
echo "Approximate duration of the auto retrain all job if you have 10 Training pods: $DURATION_10 seconds"
echo "Approximate duration of the auto retrain all job if you have 15 Training pods: $DURATION_15 seconds"
```

For other versions:

```
#!/bin/bash
calculate_duration() {
   local input_variable="$1"
   DURATION=$(("$NUM_OF_WORKSPACES_TO_TRAIN"*60 / (input_variable * 2) + "$NUM_OF_WORKSPACES_TO_TRAIN" * 2))
 }

export PROJECT_CPD_INST_OPERANDS=<namespace where Assistant is installed>

NUM_OF_WORKSPACES_TO_TRAIN=$(oc exec wa-etcd-0 -n ${PROJECT_CPD_INST_OPERANDS} -- bash -c '
 password="$( cat /var/run/credentials/pass.key )"
 etcdctl_user="root:$password"
 export ETCDCTL_USER="$etcdctl_user"

 ETCDCTL_API=3 etcdctl --cert=/etc/etcdtls/operator/etcd-tls/etcd-client.crt --key=/etc/etcdtls/operator/etcd-tls/etcd-client.key --cacert=/etc/etcdtls/operator/etcd-tls/etcd-client-ca.crt --endpoints=https://$(hostname).${CLUSTER_NAME}.cpd.svc.cluster.local:2379 get  --prefix  /bluegoat/voyager-nlu/voyager-nlu-slot-wa/workspaces/ --keys-only | sed '/^$/d' | wc -l')

echo "Number of workspaces to train $NUM_OF_WORKSPACES_TO_TRAIN"

calculate_duration 5
DURATION_5=$DURATION

calculate_duration 10
DURATION_10=$DURATION

calculate_duration 15
DURATION_15=$DURATION
echo "Approximate duration of the auto retrain all job if you have 5 Training pods: $DURATION_5 seconds"
echo "Approximate duration of the auto retrain all job if you have 10 Training pods: $DURATION_10 seconds"
echo "Approximate duration of the auto retrain all job if you have 15 Training pods: $DURATION_15 seconds"
```

In addition, you can plan to speed up the auto-retrain-all job after you get the estimation of duration. For more information about speeding up the auto-retrain-all job, see the Speeding up the auto-retrain-all job topic.

Procedure

To retrain your backend model by using the auto-retrain-all job, you do the following steps:

- Set up the environment variables for the auto-retrain-all job
- Run the auto-retrain-all job
- Validate the auto-retrain-all job

Set up the environment variables for the auto-retrain-all job

Set up the following environment variable before you run the auto-retrain-all job:

1. Set the `AUTO_RETRAIN` environment variable to `false` to disable any existing auto-retrain job:

   ```
   export AUTO_RETRAIN="false"
   ```

2. To set up the `BATCH_RETRAIN_ALL_SIZE` environment variable, you multiply the number of available training replicas, `CLU_TRAINING_REPLICAS`, with `2` based on the assumption that each model takes approximately `~30 seconds` to train a model. Use the following command to set up `BATCH_RETRAIN_ALL_SIZE`:

   ```
   export BATCH_RETRAIN_ALL_SIZE=$(($(oc get deploy ${INSTANCE}-clu-training --template='{{index .spec.replicas}}') * 2))
   ```

3. Set `WAIT_TIME_BETWEEN_BATCH_RETRAIN_IN_SECONDS_FOR_RETRAIN_ALL` to `(60-${BATCH_RETRAIN_ALL_SIZE})`:

   ```
   export WAIT_TIME_BETWEEN_BATCH_RETRAIN_IN_SECONDS_FOR_RETRAIN_ALL=$((60-${BATCH_RETRAIN_ALL_SIZE}))
   ```

4. Set `WAIT_TIME_BETWEEN_TRAININGS_FOR_RETRAIN_ALL` to 1:

   ```
   export WAIT_TIME_BETWEEN_TRAININGS_FOR_RETRAIN_ALL=1
   ```

5. Set `AUTO_RETRAIN_ALL_CRON_SCHEDULE` to the time that you want to run the auto-retrain-all job:

   ```
   export AUTO_RETRAIN_ALL_CRON_SCHEDULE=<value of cron schedule>
   ```

   For example, you can give a value such as `"0 40 19 11 3 ? 2024"`, which is in the following format:

   `(Seconds) (Minutes) (Hours) (Day of Month) (Month) (Day of Week) (Year)`

   ⚠ **Important:** You must set the time in UTC time zone.

6. Set `AUTO_RETRAIN_ALL_ENABLED` to true:

   ```
   export AUTO_RETRAIN_ALL_ENABLED="true"
   ```

Run the auto-retrain-all job

1. To run the auto-retrain-all job, use the following command:

```
export PROJECT_CPD_INST_OPERANDS=<namespace where Assistant is installed>
export INSTANCE=`oc get wa -n ${PROJECT_CPD_INST_OPERANDS} |grep -v NAME| awk '{print $1}'`

cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: ${INSTANCE}-store-admin-env-vars
  namespace: ${PROJECT_CPD_INST_OPERANDS}
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantStore
  name: ${INSTANCE}
  patchType: patchStrategicMerge
  patch:
    store-admin:
      deployment:
        spec:
          template:
            spec:
              containers:
              - name: store-admin
                env:
                - name: AUTO_RETRAIN
                  value: "${AUTO_RETRAIN}"
                - name: AUTO_RETRAIN_ALL_CRON_SCHEDULE
                  value: "${AUTO_RETRAIN_ALL_CRON_SCHEDULE}"
```

```
            - name: AUTO_RETRAIN_ALL_ENABLED
              value: "${AUTO_RETRAIN_ALL_ENABLED}"
            - name: BATCH_RETRAIN_ALL_SIZE
              value: "${BATCH_RETRAIN_ALL_SIZE}"
            - name: WAIT_TIME_BETWEEN_BATCH_RETRAIN_IN_SECONDS_FOR_RETRAIN_ALL
              value: "${WAIT_TIME_BETWEEN_BATCH_RETRAIN_IN_SECONDS_FOR_RETRAIN_ALL}"
            - name: WAIT_TIME_BETWEEN_TRAININGS_FOR_RETRAIN_ALL
              value: "${WAIT_TIME_BETWEEN_TRAININGS_FOR_RETRAIN_ALL}"
  EOF
```

Only for version 5.1.0, 5.0.3, 4.8.8, and all future versions:

2. Get the etcd endpoint by running the following:

```
oc get secret wa-cluruntime-datastore-connection-strings -o jsonpath="{.data.etcd}" | base64 --decode | jq -r '.endpoints
```

3. After you complete the auto-retrain-all job, you must disable the auto-retrain-all flag and enable auto-retrain flag by using the following commands:

```
  oc patch temporarypatch ${INSTANCE}-store-admin-env-vars -p '{"metadata":{"finalizers":[]}}' --type=merge -n ${PROJECT_CPD_INST_OPERANDS}
  oc delete temporarypatch ${INSTANCE}-store-admin-env-vars -n ${PROJECT_CPD_INST_OPERANDS}
  oc patch watsonassistantstore/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oppy.ibm.com/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
```

Validate the auto-retrain-all job

You can validate the successful completion of the auto-retrain-all job by comparing the number of `Affected workspaces found` with the `Retrained Total` count in the store-admin service log. To get the number of `Affected workspaces found` and the `Retrained Total`, run the following command:

```
oc logs $(oc get pod -l component=store-admin --no-headers -n ${PROJECT_CPD_INST_OPERANDS}  |awk '{print $1}') | grep "\[RETRAIN-ALL-SUMMARY\] Affected
workspaces found"
```

If the auto-retrain-all job is successful, the `Retrained Total` count equals the number of `Affected workspaces found`. In addition, if the difference between the counts of the `Retrained Total` and `Affected workspaces found` is small, the auto-retrain-all job completes successfully by training the remaining models in the background. However, if there is a big difference between `Retrained Total` and `Affected workspaces found`, you must look at the store-admin logs to analyze the issue and consider speeding up the auto-retrain-all job.

Speeding up the auto-retrain-all job

The duration to complete the auto-retrain-all job depends on the number of models to train. Therefore, to speed up the training process, you must `scale` the number of `CLU_TRAINING_REPLICAS` and its dependencies. For example, if you `scale` the number of `CLU_TRAINING_REPLICAS` to `x`, you must `scale` the number of dependent replicas per the following calculation:

- `TFMM_REPLICAS` to 0.5x
- `DRAGONFLY_CLU_MM_REPLICAS` to 0.3x
- `CLU_EMBEDDING_REPLICAS` to 0.2x
- `CLU_TRITON_SERVING_REPLICAS` to 0.2x.

Tip: If your calculation result for the number of models is a decimal number, then you must round-up the result to the next greater whole number. For example, if the number of `TFMM_REPLICAS` is 2.4, then round-up the value to 3.

Use the following steps to `scale` the number of models:

1. Register the values of the number of replicas per your calculation:

```
export CLU_TRAINING_REPLICAS=<value from calculation>
export TFMM_REPLICAS=<value from calculation>
export DRAGONFLY_CLU_MM_REPLICAS=<value from calculation>
export CLU_EMBEDDING_REPLICAS=<value from calculation>
export CLU_TRITON_SERVING_REPLICAS=<value from calculation>
```

2. Increase the number of `REPLICAS` by using the following command:

```
export PROJECT_CPD_INST_OPERANDS=<namespace where Assistant is installed>
export INSTANCE=`oc get wa -n ${PROJECT_CPD_INST_OPERANDS} |grep -v NAME| awk '{print $1}'`
```

```
cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: ${INSTANCE}-clu-training-replicas
  namespace: ${PROJECT_CPD_INST_OPERANDS}
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantCluTraining
  name: $INSTANCE
  patchType: patchStrategicMerge
  patch:
    clu-training:
      deployment:
        training:
          spec:
            replicas: ${CLU_TRAINING_REPLICAS}
EOF

cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: ${INSTANCE}-clu-runtime-replicas
  namespace: ${PROJECT_CPD_INST_OPERANDS}
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantCluRuntime
  name: ${INSTANCE}
  patchType: patchStrategicMerge
  patch:
    tfmm:
      deployment:
        spec:
          replicas: ${TFMM_REPLICAS}
    dragonfly-clu-mm:
      deployment:
        spec:
          replicas: ${DRAGONFLY_CLU_MM_REPLICAS}
EOF

cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: ${INSTANCE}-clu-replicas
  namespace: ${PROJECT_CPD_INST_OPERANDS}
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantClu
  name: ${INSTANCE}
  patchType: patchStrategicMerge
  patch:
    clu-embedding:
      deployment:
        spec:
          replicas: ${CLU_EMBEDDING_REPLICAS}
    clu-triton-serving:
      deployment:
        spec:
          replicas: ${CLU_TRITON_SERVING_REPLICAS}
EOF
```

3. After you complete the auto-retrain-all job, you must revert the number of `REPLICAS` to the original numbers:

```
oc patch temporarypatch ${INSTANCE}-clu-training-replicas -p '{"metadata":{"finalizers":[]}}' --type=merge -n ${PROJECT_CPD_INST_OPERANDS}
oc patch temporarypatch ${INSTANCE}-clu-runtime-replicas -p '{"metadata":{"finalizers":[]}}' --type=merge -n ${PROJECT_CPD_INST_OPERANDS}
oc patch temporarypatch ${INSTANCE}-clu-replicas -p '{"metadata":{"finalizers":[]}}' --type=merge -n ${PROJECT_CPD_INST_OPERANDS}

oc delete temporarypatch ${INSTANCE}-clu-training-replicas -n ${PROJECT_CPD_INST_OPERANDS}
```

```
 oc delete temporarypatch ${INSTANCE}-clu-runtime-replicas -n ${PROJECT_CPD_INST_OPERANDS}
 oc delete temporarypatch ${INSTANCE}-clu-replicas -n ${PROJECT_CPD_INST_OPERANDS}

 oc patch watsonassistantclutraining/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oppy.ibm.com/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
 oc patch watsonassistantcluruntime/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oppy.ibm.com/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
 oc patch watsonassistantclu/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oppy.ibm.com/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
 oc patch watsonassistantclutraining/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oper8.org/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
 oc patch watsonassistantcluruntime/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oper8.org/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
 oc patch watsonassistantclu/${INSTANCE} -p "{\"metadata\":{\"annotations\":{\"oper8.org/temporary-patches\":null}}}" --type=merge -n
${PROJECT_CPD_INST_OPERANDS}
```

Troubleshooting known issues for IBM Cloud Pak for Data

IBM Cloud Pak for Data

Get help with solving issues that you might encounter with Watson Assistant on IBM Cloud Pak for Data.

4.5.x

Pod `RESTARTS` count stays at 0 after a 4.5.x upgrade even though a few assistant pods are restarting

- Problem: After you upgrade Watson Assistant, the pod `RESTARTS` count stays at 0 even though certain assistant pods are restarting.
- Cause: During the upgrade, custom resources that are owned by Watson Assistant for the `certificates.certmanager.k8s.io` CRD are deleted by using a script that runs in the background. Sometimes the CR deletion script completes before the assistant operator gets upgraded. In that case, the old assistant operator might re-create custom resources for the `certificates.certmanager.k8s.io` CRD. Leftover CRs might cause the certificate manager to continuously regenerate some certificate secrets, causing some assistant pods to restart recursively.
- Solution: Run the following script to delete leftover custom resources for the `certificates.certmanager.k8s.io` CRD after you set INSTANCE (normally `wa`) and PROJECT_CPD_INSTANCE variables:

```
for i in `oc get certificates.certmanager.k8s.io -l icpdsupport/addOnId=assistant --namespace ${PROJECT_CPD_INSTANCE} | grep "${INSTANCE}-"| awk '{print $1}'`; do
oc delete certificates.certmanager.k8s.io $i --namespace ${PROJECT_CPD_INSTANCE}; done
```

4.5.0

Data Governor not healthy after installation

After Watson Assistant is installed, the `dataexhausttenant` custom resource that is named `wa-data-governor-ibm-data-governor-data-exhaust-internal` gets stuck in the `Topics` phase. When this happens, errors in the Data Governor pods report that the service does not exist.

1. Get the status of the `wa-data-governor` custom resource:

```
oc get DataExhaust
```

2. Wait for the `wa-data-governor` custom resource to be in the `Completed` phase:

```
NAME              STATUS      VERSION   COMPLETED
wa-data-governor  Completed   master    1s
```

3. Pause the reconciliation of the `wa-data-governor` custom resource:

```
oc patch dataexhaust wa-data-governor -p '{"metadata":{"annotations":{"pause-reconciliation":"true"}}}' --type merge
```

4. Apply the fix to the `dataexhausttenant` custom resource:

```
oc patch dataexhausttenant wa-data-governor-ibm-data-governor-data-exhaust-internal  -p '{"spec":{"topics":{"data":{"replicas": 1}}}}' --type merge
```

5. Wait for the Data Governor pods to stop failing. You can restart the admin pods to speed up this process.
6. Continue the reconciliation of the `wa-data-governor` custom resource:

```
oc patch dataexhaust wa-data-governor --type=json -p='[{"op": "remove", "path": "/metadata/annotations/pause-reconciliation"}]'
```

RabbitMQ gets stuck in a loop after several installation attempts

After an initial installation or upgrade failure and repeated attempts to retry, the common services RabbitMQ operator pod can get into a `CrashLoopBackOff` state. For example, the log might include the following types of messages:

```
"error":"failed to upgrade release: post-upgrade hooks failed: warning:
Hook post-upgrade ibm-rabbitmq/templates/rabbitmq-backup-labeling-job.yaml
failed: jobs.batch "{%name}-ibm-rabbitmq-backup-label" already exists"
```

Resources for the `ibm-rabbitmq-operator.v1.0.11` component must be removed before a new installation or upgrade is started. If too many attempts occur in succession, remaining resources can cause new installations to fail.

1. Delete the RabbitMQ backup label job from the previous installation or upgrade attempt. Look for the name of the job in the logs. The name ends in `-ibm-rabbitmq-backup-label`:

   ```
   oc delete job {%name}-ibm-rabbitmq-backup-label -n ${PROJECT_CPD_INSTANCE}
   ```

2. Check that the pod returns a `Ready` state:

   ```
   oc get pods -n ibm-common-services | grep ibm-rabbitmq
   ```

Preparing to install a size large deployment

If you specify `large` for the `watson_assistant_size` option when you install Watson Assistant, the installation fails to complete successfully.

Before you install a size large deployment of Watson Assistant, apply the following fix. The following fix uses `wa` as the name of the Watson Assistant instance and `cpd` as the namespace where Watson Assistant is installed. These values are set in environment variables. Before you run the command, update the `INSTANCE` variable with the name of your instance, and update the `NAMESPACE` variable with the namespace where your instance in installed:

```
INSTANCE=wa ; \
NAMESPACE=cpd ; \
#DRY_RUN="--dry-run=client --output=yaml"     # To apply the changes, change to an empty string
DRY_RUN=""
cat <<EOF | tee wa-network-policy-base.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  annotations:
    oppy.ibm.com/internal-name: infra.networkpolicy
  labels:
    app: ${INSTANCE}-network-policy
    app.kubernetes.io/instance: ${INSTANCE}
    app.kubernetes.io/managed-by: Ansible
    app.kubernetes.io/name: watson-assistant
    component: network-policy
    icpdsupport/addOnId: assistant
    icpdsupport/app: ${INSTANCE}-network-policy
    icpdsupport/ignore-on-nd-backup: "true"
    icpdsupport/serviceInstanceId: inst-1
    service: conversation
    slot: ${INSTANCE}
    tenant: PRIVATE
    velero.io/exclude-from-backup: "true"
  name: ${INSTANCE}-network-policy
  namespace: ${NAMESPACE}
spec:
  ingress:
  - from:
    - podSelector:
        matchLabels:
          service: conversation
          slot: ${INSTANCE}
    - podSelector:
        matchLabels:
```

```
      slot: global
- podSelector:
    matchLabels:
      component: watson-gateway
- podSelector:
    matchLabels:
      component: dvt
- podSelector:
    matchLabels:
      dwf_service: ${INSTANCE}-clu
      network-policy: allow-egress
- podSelector:
    matchLabels:
      app: 0020-zen-base
- namespaceSelector:
    matchLabels:
      ns: ${NAMESPACE}
  podSelector:
    matchLabels:
      app: 0020-zen-base
- podSelector:
    matchLabels:
      component: ibm-nginx
- namespaceSelector:
    matchLabels:
      ns: ${NAMESPACE}
  podSelector:
    matchLabels:
      component: ibm-nginx
- namespaceSelector:
    matchLabels:
      assistant.watson.ibm.com/role: operator
  podSelector:
    matchLabels:
      release: assistant-operator
- namespaceSelector:
    matchLabels:
      assistant.watson.ibm.com/role: operator
  podSelector:
    matchLabels:
      app: watson-assistant-operator
- namespaceSelector:
    matchLabels:
      assistant.watson.ibm.com/role: operator
  podSelector:
    matchLabels:
      app.kubernetes.io/instance: watson-assistant-operator
- namespaceSelector:
    matchLabels:
      assistant.watson.ibm.com/role: operator
    podSelector:
      matchLabels:
        app.kubernetes.io/instance: ibm-etcd-operator-release
  - namespaceSelector:
      matchLabels:
        assistant.watson.ibm.com/role: operator
    podSelector:
      matchLabels:
        app.kubernetes.io/instance: ibm-etcd-operator
  podSelector:
    matchLabels:
      service: conversation
      slot: ${INSTANCE}
  policyTypes:
  - Ingress
EOF
for MICROSERVICE in analytics clu-embedding clu-serving clu-training create-slot-job data-governor dialog dragonfly-clu-mm ed es-store etcd integrations master nlu
recommends sireg-ubi-ja-tok-20160902 sireg-ubi-ko-tok-20181109 spellchecker-mm store store-admin store-cronjob store-sync store_db_creator store_db_schema_updater
system-entities tfmm ui ${INSTANCE}-redis webhooks-connector
do
  # Change name and add component to selector
```

```
  # Apply to the cluster
  cat wa-network-policy-base.yaml | \
    oc patch --dry-run=client --output=yaml -f - --type=merge --patch "{
        \"metadata\": {\"name\": \"${INSTANCE}-network-policy-$( echo $MICROSERVICE | tr _ -)\"},
        \"spec\": {\"podSelector\":{\"matchLabels\":{\"component\": \"${MICROSERVICE}\"}}}
      }" |
    oc apply -f - ${DRY_RUN}
done
```

Fixing a size large installation

Apply this fix if you installed a size `large` deployment, and your installation fails to complete successfully. In some cases, pods aren't able to communicate with other pods, and the Transmission Control Protocol (TCP) connections can't be established.

To confirm whether you are affected by this issue, run the following command:

```
oc logs --selector app=sdn --namespace openshift-sdn --container sdn | grep "Ignoring NetworkPolicy"
```

If you are affected, you see output similar to the following example:

```
W0624 12:58:21.407901    2480 networkpolicy.go:484] Ignoring NetworkPolicy cpd/wa-network-policy because it generates
```

If you encounter this error, apply the following fix to resolve the issue. The following fix uses `wa` as the name of the Watson Assistant instance. This value is set in an environment variable. Before you run the command, update the `INSTANCE` variable with the name of your instance:

```
INSTANCE=wa ; \
#DRY_RUN="--dry-run=client --output=yaml"     # To apply the changes, change to an empty string
DRY_RUN=""
for MICROSERVICE in analytics clu-embedding clu-serving clu-training create-slot-job data-governor dialog dragonfly-clu-mm ed es-store etcd integrations master nlu
recommends sireg-ubi-ja-tok-20160902 sireg-ubi-ko-tok-20181109 spellchecker-mm store store-admin store-cronjob store-sync store_db_creator store_db_schema_updater
system-entities tfmm ui ${INSTANCE}-redis webhooks-connector
do
  # Get original networking policy
  # Clean up metadata fields to get the resource applied by Watson Assistant
  # Change name and add component to selector
  # Apply to the cluster
  oc get networkpolicy $INSTANCE-network-policy --output yaml | \
  oc patch --dry-run=client --output=yaml -f - --type=json --patch='[
      {"op":"remove", "path":"/metadata/creationTimestamp"},
      {"op":"remove", "path":"/metadata/generation"},
      {"op":"remove", "path":"/metadata/resourceVersion"},
      {"op":"remove", "path":"/metadata/uid"},
      {"op":"remove", "path":"/metadata/annotations/kubectl.kubernetes.io~1last-applied-configuration"},
      {"op":"remove", "path":"/metadata/ownerReferences"}
    ]' | \
    oc patch --dry-run=client --output=yaml -f - --type=merge --patch "{
        \"metadata\": {\"name\": \"${INSTANCE}-network-policy-$( echo $MICROSERVICE | tr _ -)\"},
        \"spec\": {\"podSelector\":{\"matchLabels\":{\"component\": \"${MICROSERVICE}\"}}}
      }" |
    oc apply -f - ${DRY_RUN}
done
```

Unable to scale the size of Redis pods

When you scale the deployment size of Watson Assistant, the Redis pods do not scale correctly and don't match the new size of the deployment (`small`, `medium`, or `large`). This problem is a known issue in Redis operator v1.5.1.

When you scale the size of your deployment, you must delete the Redis custom resource. Redis automatically re-creates the custom resource with the correct size and pods.

To delete and re-create the Redis custom resource:

1. Source the environment variables from the script:

   ```
   source ./cpd_vars.sh
   ```

   If you don't have the script that defines the environment variables, see Setting up installation environment variables.

2. Export the name of the Redis custom resource to an environment variable:

```
export REDIS_CR_NAME=`oc get redissentinels.redis.databases.cloud.ibm.com -l icpdsupport/addOnId=assistant -n ${PROJECT_CPD_INSTANCE} | grep -v NAME| awk '{print $1}'`
```

3. Delete the Redis custom resource:

```
oc delete redissentinels.redis.databases.cloud.ibm.com ${REDIS_CR_NAME} -n ${PROJECT_CPD_INSTANCE}
```

   It might take approximately 5 minutes for the custom resource to be re-created.

4. Verify that Redis is running:

```
oc get redissentinels.redis.databases.cloud.ibm.com -n ${PROJECT_CPD_INSTANCE}
```

5. Export the name of your instance as an environment variable:

```
export INSTANCE=`oc get wa -n ${PROJECT_CPD_INSTANCE} |grep -v NAME| awk '{print $1}'`
```

6. Delete the Redis analytics secret:

```
oc delete secrets ${INSTANCE}-analytics-redis
```

7. Delete the `${INSTANCE}-analytics` deployment pods.

4.0.x

Data Governor error causes deployment failure

The following fix applies to 4.0.0 through 4.0.8. In some cases, the deployment is stuck and pods are not coming up because of an issue with the interaction between the Events operator and the Data Governor custom resource (CR).

Complete the following steps to determine whether you are impacted by this issue and, if necessary, apply the patch to resolve it:

1. To determine whether you are impacted, run the following command to see whether the CR was applied successfully:

```
oc get dataexhaust wa-data-governor -n $OPERAND_NS -o yaml
```

   If you do not receive any error, then you do not need to apply the patch. If you receive an error similar to the following example, complete the next step to apply the patch:

```
message: 'Failed to create object: b"{"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"Internal
    error occurred: replace operation does not apply: doc is missing path: /metadata/labels/icpdsupport/serviceInstanceId:
    missing value","reason":"InternalError","details":{"causes":[{"message":"replace
    operation does not apply: doc is missing path: /metadata/labels/icpdsupport/serviceInstanceId:
    missing value"}]},"code":500}\n'"
  reason: Failed
  status: "True"
  type: Failure
ibmDataGovernorService: InProgress
```

2. From your operand namespace, run the following command to apply the patch. In the command, `wa` is used as the name of the instance. Replace this value with the name of your instance:

```
cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: wa-data-governor
spec:
   apiVersion: assistant.watson.ibm.com/v1
   kind: WatsonAssistant
   name: wa     # Replace wa with the name of your Watson Assistant instance
   patch:
     data-governor:
       dataexhaust:
```

```
        spec:
          additionalLabels:
            icpdsupport/serviceInstanceId: inst-1
      kafkauser:
        metadata:
          labels:
            icpdsupport/serviceInstanceId: inst-1
patchType: patchStrategicMerge
EOF
```

Wait about 15 minutes for the changes to take effect.

3. Validate that the patch was applied successfully:

```
oc get dataexhaust wa-data-governor -n $OPERAND_NS -o yaml
```

The patch was applied successfully when the value of `serviceInstanceId` is `inst-1`:

```
spec:
  additionalLabels:
    icpdsupport/serviceInstanceId: inst-1
```

Security context constraint permission errors

The following fix applies to 4.0.0 through 4.0.5. If a cluster has a security context constraint (SCC) that takes precedence over **restricted** SCCs and has different permissions than **restricted** SCCs, then 4.0.0 through 4.0.5 installations might fail with permission errors. For example, the `update-schema-store-db-job` job reports errors similar to the following example:

```
oc logs wa-4.0.2-update-schema-store-db-job-bpsdr postgres-is-prepared
Waiting until postgres is running and responding
psql: error: could not read root certificate file "/tls/ca.crt": Permission denied
    - The basic command to postgres failed (retry in 5 sec)
psql: error: could not read root certificate file "/tls/ca.crt": Permission denied
    - The basic command to postgres failed (retry in 5 sec)
psql: error: could not read root certificate file "/tls/ca.crt": Permission denied
    - The basic command to postgres failed (retry in 5 sec)
..
..
```

Other pods might have similar permission errors. If you look at the SCCs of the pods, you can see they are not restricted. For example, if you run the `oc describe pod wa-etcd-0 |grep scc` command, you get an output similar to the following example:

```
openshift.io/scc: fsgroup-scc
```

To fix this issue, raise the priority of the **restricted** SCC so that it takes precedence:

1. Run the following command:

```
oc edit scc restricted
```

2. Change the `priority` from `null` to `1`.

Now, new pods default back to the expected **restricted** SCC. When you run the `oc describe pod wa-etcd-0 |grep scc` command, you get an output similar to the following example:

```
openshift.io/scc: restricted
```

Unable to collect logs with a webhook

The following fix applies to all versions of Watson Assistant 4.0.x. If you're unable to collect logs with a webhook, it might be because you are using a webhook that connects to a server that is using a self-signed certificate. If so, complete the following steps to import the certificate into the keystore so that you can collect logs with a webhook:

1. Log in to cluster and `oc project cpd-instance`, which is the namespace where the instance is located.
2. Run the following command. In the following command, replace `INSTANCE_NAME` with the name of your instance and replace

`CUSTOM_CERTIFICATE` with your Base64 encoded custom certificate key:

```
INSTANCE="INSTANCE_NAME"     # Replace INSTANCE_NAME with the name of the Watson Assistant instance
CERT="CUSTOM_CERTIFICATE"     # Replace CUSTOM_CERTIFICATE with the custom certificate key

cat <<EOF | oc apply -f -
apiVersion: v1
data:
  ca_cert: ${CERT}
kind: Secret
metadata:
  name: ${INSTANCE}-custom-webhooks-cert
type: Opaque
---
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: ${INSTANCE}-add-custom-webhooks-cert
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantStore
  name: ${INSTANCE}
  patchType: patchStrategicMerge
  patch:
    webhooks-connector:
      deployment:
        spec:
          template:
            spec:
              containers:
              - name: webhooks-connector
                env:
                - name: CERTIFICATES_IMPORT_LIST
                  value: /etc/secrets/kafka/ca.pem:kafka_ca,/etc/secrets/custom/ca.pem:custom_ca
                volumeMounts:
                - mountPath: /etc/secrets/custom
                  name: custom-cert
                  readOnly: true
              volumes:
              - name: custom-cert
                secret:
                  defaultMode: 420
                  items:
                  - key: ca_cert
                    path: ca.pem
                  secretName: ${INSTANCE}-custom-webhooks-cert
EOF
```

3. Wait approximately 10 minutes for the `wa-webhooks-connector` pod to restart. This pod restarts automatically.
4. After the pod restarts, check the logs by running the following command. In the command, replace `XXXX` with the suffix of the `wa-webhooks-connector` pod:

   ```
   oc logs wa-webhooks-connector-XXXX     // Replace XXXX with the suffix of the wa-webhooks-connector pod
   ```

   After you run this command, you should see two lines similar to the following example at the beginning of the log:

   ```
   Certificate was added to keystore
   Certificate was added to keystore
   ```

   When you see these two lines, then the custom certificate was properly imported into the keystore.

4.0.5

Install Redis if foundational services version is higher than 3.14.1

If you are installing the Redis operator with an IBM Cloud Pak foundational services version higher than 3.14.1, the Redis operator might get stuck in `Pending` status. If you have an air-gapped cluster, complete the steps in the Air-gapped cluster section to resolve this issue. If you are using the IBM

Entitled Registry, complete the steps in the IBM Entitled Registry section to resolve this issue.

Air-gapped cluster

1. Check the status of the Redis operator:

```
oc get opreq common-service-redis -n ibm-common-services -o jsonpath='{.status.phase}  {"\n"}'
```

2. If the Redis operand request is stuck in `Pending` status, delete the operand request:

```
oc delete opreq watson-assistant-redis -n ibm-common-services
```

3. Set up your environment to download the CASE packages.

    a. Create the directories where you want to store the CASE packages:

    ```
    mkdir -p $HOME/offline/cpd
    mkdir -p $HOME/offline/cpfs
    ```

    b. Set the following environment variables:

    ```
    export CASE_REPO_PATH=https://github.com/IBM/cloud-pak/raw/master/repo/case
    export OFFLINEDIR=$HOME/offline/cpd
    export OFFLINEDIR_CPFS=$HOME/offline/cpfs
    ```

4. Download the Redis operator and IBM Cloud Pak® for Data platform operator CASE packages:

```
cloudctl case save \
--repo ${CASE_REPO_PATH} \
--case ibm-cloud-databases-redis \
--version  1.4.5  \
--outputdir $OFFLINEDIR

cloudctl case save \
--repo ${CASE_REPO_PATH} \
--case ibm-cp-datacore \
--version 2.0.10 \
--outputdir ${OFFLINEDIR} \
--no-dependency
```

5. Create the Redis catalog source:

```
cloudctl case launch \
--case ${OFFLINEDIR}/ibm-cloud-databases-redis-1.4.5.tgz \
--inventory redisOperator \
--action install-catalog \
--namespace openshift-marketplace \
--args "--registry icr.io --inputDir ${OFFLINEDIR} --recursive"
```

6. Set the environment variables for your registry credentials:

```
export PRIVATE_REGISTRY_USER=username
export PRIVATE_REGISTRY_PASSWORD=password
export PRIVATE_REGISTRY={registry-info}
```

7. Run the following command to store the credentials:

```
cloudctl case launch \
--case ${OFFLINEDIR}/ibm-cp-datacore-2.0.10.tgz \
--inventory cpdPlatformOperator \
--action configure-creds-airgap \
--args "--registry ${PRIVATE_REGISTRY} --user ${PRIVATE_REGISTRY_USER} --pass ${PRIVATE_REGISTRY_PASSWORD}"
```

8. Mirror the images:

```
export USE_SKOPEO=true
```

```
cloudctl case launch \
--case ${OFFLINEDIR}/ibm-cp-datacore-2.0.10.tgz \
--inventory cpdPlatformOperator \
--action mirror-images \
--args "--registry ${PRIVATE_REGISTRY} --user ${PRIVATE_REGISTRY_USER} --pass ${PRIVATE_REGISTRY_PASSWORD} --inputDir ${OFFLINEDIR}"
```

9. Create the Redis subscription.

    a. Export the project that contains the IBM Cloud Pak® for Data operator:

    ```
    export OPERATOR_NS=ibm-common-services|cpd-operators     # Select the project that contains the Cloud Pak for Data operator
    ```

    b. Create the subscription:

    ```
    cat <<EOF | oc apply --namespace $OPERATOR_NS -f -
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: ibm-cloud-databases-redis-operator
    spec:
      name: ibm-cloud-databases-redis-operator
      source: ibm-cloud-databases-redis-operator-catalog
      sourceNamespace: openshift-marketplace
    EOF
    ```

IBM Entitled Registry

1. Check the status of the Redis operator:

    ```
    oc get opreq common-service-redis -n ibm-common-services -o jsonpath='{.status.phase}  {"\n"}'
    ```

2. If the Redis operand request is stuck in `Pending` status, delete the operand request:

    ```
    oc delete opreq watson-assistant-redis -n ibm-common-services
    ```

3. Create the Redis subscription. Create one of the following two subscriptions, depending on whether you are using.

    a. Export the project that contains the IBM Cloud Pak® for Data operator:

    ```
    export OPERATOR_NS=ibm-common-services|cpd-operators     # Select the project that contains the Cloud Pak for Data operator
    ```

    b. Create the subscription. Choose one of the following two subscriptions, depending on how you are using the IBM Entitled Registry:

    - IBM Entitled Registry from the `ibm-operator-catalog`:

    ```
    cat <<EOF | oc apply --namespace  $OPERATOR_NS -f -
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: ibm-cloud-databases-redis-operator
    spec:
      name: ibm-cloud-databases-redis-operator
      source: ibm-operator-catalog
      sourceNamespace: openshift-marketplace
    EOF
    ```

    - IBM Entitled Registry with catalog sources that pull specific versions of the images:

    ```
    cat <<EOF | oc apply --namespace $OPERATOR_NS -f -
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: ibm-cloud-databases-redis-operator
    spec:
      name: ibm-cloud-databases-redis-operator
      source: ibm-cloud-databases-redis-operator-catalog
    ```

```
  sourceNamespace: openshift-marketplace
EOF
```

4. Validate that the operator was successfully created.

    a. Run the following command to confirm that the subscription was applied:

    ```
    oc get sub -n $OPERATOR_NS  ibm-cloud-databases-redis-operator -o jsonpath='{.status.installedCSV} {"\n"}'
    ```

    b. Run the following command to confirm that the operator is installed:

    ```
    oc get pod -n $OPERATOR_NS -l app.kubernetes.io/name=ibm-cloud-databases-redis-operator \
    -o jsonpath='{.items[0].status.phase} {"\n"}'
    ```

4.0.4

Integrations image problem on air-gapped installations

If your installation is air-gapped, your integrations image fails to properly start.

If the installation uses the IBM Entitled Registry to pull images, complete the steps in the **IBM Entitled Registry** section. If the installation uses a private Docker registry to pull images, complete the steps in the **Private Docker registry** section.

IBM Entitled Registry

If your installation uses the IBM Entitled Registry to pull images, complete the following steps to add an override entry to the CR:

1. Get the name of your instance by running the following command:

    ```
    oc get wa
    ```

2. Edit and save the CR.

    a. Run the following command to edit the CR. In the command, replace `INSTANCE_NAME` with the name of the instance:

    ```
    oc edit wa INSTANCE_NAME
    ```

    b. Edit the CR by adding the following lines:

    ```
    appConfigOverrides:
      container_images:
      integrations:
        image: cp.icr.io/cp/watson-assistant/servicedesk-integration
        tag: 20220106-143142-0ea3fbf7-wa_icp_4.0.5-signed@sha256:7078fdba4ab0b69dbb93f47836fd9fcb7cfb12f103662fef0d9d1058d2553910
    ```

3. Wait for the operator to pick up the change and start a new integrations pod. This might take up to 10 minutes.

4. After the new integrations pod starts, the old pod terminates. When the new pod starts, the server starts locally and the log looks similar to the following example:

    ```
    oc logs -f ${INTEGRATIONS_POD}
    [2022-01-07T01:33:13.609] [OPTIMIZED] db.redis.RedisManager - Redis trying to connect. counter# 1
    [2022-01-07T01:33:13.628] [OPTIMIZED] db.redis.RedisManager - Redis connected
    [2022-01-07T01:33:13.629] [OPTIMIZED] db.redis.RedisManager - Redis is ready to serve!
    [2022-01-07T01:33:14.614] [OPTIMIZED] Server - Server started at: https://localhost:9449
    ```

Private Docker registry

If your installation uses a private Docker registry to pull images, complete the following steps to download and push the new integrations image to your private Docker registry and add an override entry to the CR:

1. Edit the CSV file to add the new integrations image.

    a. Run the following command to open the CSV file:

    ```
    vi $OFFLINEDIR/ibm-watson-assistant-4.0.4-images.csv
    ```

b. Add the following line to the CSV file immediately after the existing integrations image:

```
cp.icr.io,cp/watson-assistant/servicedesk-integration,20220106-143142-0ea3fbf7-wa_icp_4.0.5-signed,sha256:7078fdba4ab0b69dbb93f47836fd9fcb7cfb12f103662fef0d9d1058d2553910,IMAGE,linux,x86_64,"",0,CASE,"",ibm_wa_4_0_0;ibm_wa_4_0_2;ibm_wa_4_0_4;vLatest
```

2. Mirror the image again by using the commands that you used to download and push all the images, for example:

```
cloudctl case launch \
  --case ${OFFLINEDIR}/ibm-cp-datacore-2.0.9.tgz \
  --inventory cpdPlatformOperator \
  --action configure-creds-airgap \
  --args "--registry cp.icr.io --user cp --pass $PRD_ENTITLED_REGISTRY_APIKEY --inputDir ${OFFLINEDIR}"
```

3. Get the name of your instance by running the following command:

```
oc get wa
```

4. Edit and save the CR.

a. Run the following command to edit the CR. In the command, replace `INSTANCE_NAME` with the name of the instance:

```
oc edit wa INSTANCE_NAME
```

b. Edit the CR by adding the following lines:

```
appConfigOverrides:
  container_images:
  integrations:
    image: cp.icr.io/cp/watson-assistant/servicedesk-integration
    tag: 20220106-143142-0ea3fbf7-wa_icp_4.0.5-signed@sha256:7078fdba4ab0b69dbb93f47836fd9fcb7cfb12f103662fef0d9d1058d2553910
```

5. Wait for the operator to pick up the change and start a new integrations pod. This might take up to 10 minutes.

6. After the new integrations pod starts, the old pod terminates. When the new pod starts, the server starts locally and the log looks similar to the following example:

```
oc logs -f ${INTEGRATIONS_POD}
[2022-01-07T01:33:13.609] [OPTIMIZED] db.redis.RedisManager - Redis trying to connect. counter# 1
[2022-01-07T01:33:13.628] [OPTIMIZED] db.redis.RedisManager - Redis connected
[2022-01-07T01:33:13.629] [OPTIMIZED] db.redis.RedisManager - Redis is ready to serve!
[2022-01-07T01:33:14.614] [OPTIMIZED] Server - Server started at: https://localhost:9449
```

4.0.0

Install Watson Assistant 4.0.0 with EDB version 1.8

⚠ **Important:** Complete this task only if you need a fresh installation of Watson Assistant 4.0.0. Do not complete this task on existing clusters with data. Completing this task on an existing cluster with data results in data loss.

If you upgraded to EDB version 1.8 and need a new installation of Watson Assistant 4.0.0, complete the following steps. In the following steps, `wa` is used as the name of the custom resource. Replace this value with the name of your custom resource:

1. First, apply the following patch:

```
cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: wa-postgres-180-hotfix
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistantStore
  name: wa     # Specify the name of your custom resource
  patchType: patchStrategicMerge
```

```
  patch:
    postgres:
      postgres:
        spec:
          bootstrap:
            initdb:
              options:
               - "--encoding=UTF8"
               - "--locale=en_US.UTF-8"
               - "--auth-host=scram-sha-256"
EOF
```

2. Run the following command to check that the temporary patch is applied to the `WatsonAssistantStore` custom resource. It might take up to 10 minutes after the store custom resource is updated:

   ```
   oc get WatsonAssistantStore wa -o jsonpath='{.metadata.annotations.oppy\.ibm\.com/temporary-patches}' ; echo
   ```

   If the patch is applied, the command returns output similar to the following example:

   ```
   {"wa-postgres-180-hotfix": {"timestamp": "2021-09-23T15:48:12.071497", "api_version": "assistant.watson.ibm.com/v1"}}`
   ```

3. Run the following command to delete the Postgres instance:

   ```
   oc delete clusters.postgresql.k8s.enterprisedb.io wa-postgres
   ```

4. Wait 10 minutes. Then, run the following command to check the Postgres instance:

   ```
   oc get pods | grep wa-postgres
   ```

   When the Postgres instance is running properly, the command returns output similar to the following example:

   ```
   wa-postgres-1                                     1/1     Running     0          37m
   wa-postgres-2                                     1/1     Running     0          36m
   wa-postgres-3                                     1/1     Running     0          36m
   ```

5. Run the following command to check that the jobs that initialize the Postgres database completed successfully:

   ```
   oc get jobs wa-create-slot-store-db-job wa-4.0.0-update-schema-store-db-job
   ```

   When the jobs complete successfully, the command returns output similar to the following example:

   ```
   NAME                                   COMPLETIONS   DURATION   AGE
   wa-create-slot-store-db-job            1/1           3s         31m
   wa-4.0.0-update-schema-store-db-job    1/1           13s        31m
   ```

   If the jobs don't complete successfully, then they timed out and need to be re-created. Delete the jobs by running the `oc delete jobs wa-create-slot-store-db-job wa-4.0.0-update-schema-store-db-job` command. The jobs are re-created after 10 minutes by the operator.

For information about upgrading from Watson Assistant 4.0.0 to Watson Assistant 4.0.2, see Upgrading Watson Assistant to a newer 4.0 refresh.

1.5.0

Search skill not working because of custom certificate

The search skill, which is the integration with the IBM Watson® Discovery service, might not work if you configured a custom TLS certificate in IBM Cloud Pak® for Data. If the custom certificate is not signed by a well-known certificate authority (CA), the search skill does not work as expected. You might also see errors in the **Try out** section of the search skill.

Validate the issue

First, check the logs of the search skill pods to confirm whether this issue applies to you.

1. Run the following command to list the search skill pods:

   ```
   oc get pods -l component=skill-search
   ```

2. Run the following command to check the logs for the following exception:

   ```
   oc logs -l component=skill-search | grep "IBMCertPathBuilderException"
   ```

   The error looks similar to the following example:

   ```
   {"level":"ERROR","logger":"wa-skills","message":"Search skill exception","exception":[{"message":"com.ibm.jsse2.util.h: PKIX path building failed:
   com.ibm.security.cert.IBMCertPathBuilderException: unable to find valid certification path to requested target","name":"SSLHandshakeException"}]}
   ```

   If you see this error, continue to follow the steps to apply the fix.

Apply the fix

To fix the search skill, you inject the CA that signed your TLS certificate into the Java truststore that is used by the search skill. The search skill pods are then able to validate your certificate and communicate with the IBM Watson® Discovery service.

1. First, get your certificate. You might have this certificate, but in these steps you can retrieve the certificate directly from the cluster.

   a. Run the following command to check that the secret exists:

   ```
   oc get secret external-tls-secret
   ```

   b. Run the following command to retrieve the certificate chain from the secret:

   ```
   oc get secret external-tls-secret --output jsonpath='{.data.cert\.crt}' | base64 -d | tee ingress_cert_chain.crt
   ```

   c. Extract the CA certificate.

   The `ingress_cert_chain.crt` file typically contains multiple certificates. The last certificate in the file is usually your CA certificate.

   d. Copy the last certificate block that begins with `-----BEGIN CERTIFICATE-----` and ends with `-----END CERTIFICATE-----`.

   e. Save this certificate in the `ingress_ca.crt` file. When you save the `ingress_ca.crt` file, the `-----BEGIN CERTIFICATE-----` line must be the first line of the file, and the `-----END CERTIFICATE-----` line must be the last line of the file.

2. Retrieve the truststore that is used by the search skill pods.

   a. Run the following command to list the search skill pods:

   ```
   oc get pods -l component=skill-search
   ```

   b. Run the following command to set the `SEARCH_SKILL_POD` environment variable with the search skill pod name:

   ```
   SEARCH_SKILL_POD="$(oc get pods -l component=skill-search --output custom-columns=NAME:.metadata.name --no-headers | head -n 1
   ```

   c. Run the following command to see the selected pod:

   ```
   echo "Selected search skill pod: ${SEARCH_SKILL_POD}"
   ```

   d. Retrieve the truststore file. The `cacerts` file is the default truststore that is used by Java. It contains the list of the certificate authorities that Java trusts by default. Run the following command to copy the binary `cacerts` file from the pod into your current directory:

   ```
   oc cp ${SEARCH_SKILL_POD}:/opt/ibm/java/jre/lib/security/cacerts cacerts
   ```

3. Run the following command to inject the `ingress_ca.crt` file into the `cacerts` file:

   ```
   keytool -import -trustcacerts -keystore cacerts -storepass changeit -alias customer_ca -file ingress_ca.crt
   ```

   Tip: You can run the `keytool -list -keystore cacerts -storepass changeit | grep customer_ca -A 1` command to check that your CA certificate is included in the `cacerts` file.

4. Run the following command to create the configmap that contains the updated `cacerts` file:

   ```
   oc create configmap watson-assistant-skill-cacerts --from-file=cacerts
   ```

Because the `cacerts` file is binary, the output of the `oc describe configmap watson-assistant-skill-cacerts` command shows an empty data section. To check whether the updated `cacerts` file is present in the configmap, run the `oc get configmap watson-assistant-skill-cacerts --output yaml` command.

5. Override the `cacerts` file in the search skill pods. In this step, you configure the operator to override the `cacerts` file in the search skill pods with the updated `cacerts` file. In the following example file, the instance is called `watson-assistant---wa`. Replace this value with the name of your instance:

```
$ cat <<EOF | oc apply -f -
kind: TemporaryPatch
apiVersion: com.ibm.oppy/v1
metadata:
  name: watson-assistant---wa-skill-cert
spec:
  apiVersion: com.ibm.watson.watson-assistant/v1
  kind: WatsonAssistantSkillSearch
  name: "watson-assistant---wa"    # Replace this with the name of your Watson Assistance instance
  patchType: patchStrategicMerge
  patch:
    "skill-search":
      deployment:
        spec:
          template:
            spec:
              volumes:
               - name: updated-cacerts
                 configMap:
                   name: watson-assistant-skill-cacerts
                   defaultMode: 420
              containers:
              - name: skill-search
                volumeMounts:
                - name: updated-cacerts
                  mountPath: /opt/ibm/java/jre/lib/security/cacerts
                  subPath: cacerts
EOF
```

6. Wait until new search skill pods are created. It might take up to 10 minutes before the updates take effect.

7. Check that the search skill feature is working as expected.

Disable Horizontal Pod Autoscaling and set a maximum number of master pods

Horizontal Pod Autoscaling (HPA) is enabled automatically. As a result, the number of replicas changes dynamically in the range of 1 to 10 replicas. You can disable HPA if you want to limit the maximum number of master pods or if you're concerned about master pods being created and deleted too frequently.

1. Disable HPA for the `master` microservice by running the following command. In these steps, substitute your instance name for the `INSTANCE_NAME` variable:

```
oc patch wa ${INSTANCE_NAME} --type='json' --patch='[{"op": "add", "path": "/appConfigOverrides/clu", "value":{"master":{"autoscaling":{"enabled":false}}}}]'
```

2. Wait until the information propagates into the operator:

```
sleep 600
```

3. Run the following command to remove HPA for the `master` microservice:

```
oc delete hpa ${INSTANCE_NAME}-master
```

4. Wait for about 30 seconds:

```
sleep 30
```

5. Scale down the `master` microservice to the number of replicas that you want. In the following example, the `master` microservice is scaled down to two replicas:

```
oc scale deploy ${INSTANCE_NAME}-master --replicas=2
```

cpd-watson-assistant-1.5.0-patch-1

cpd-watson-assistant-1.5.0-patch-1 is available for installations of version 1.5.0.

This patch includes:

- Configuration changes for FIPS compatibility
- A fix for a blank screen issue that occurred when selecting multiple items in the UI
- An update to resolve a localStorage issue when developing in Google Chrome
- A fix for a scrolling issue

Resizing the Redis statefulset memory and cpu values after applying patch 1 for 1.5.0

Here are steps to resize Redis statefulset memory and cpu values after applying cpd-watson-assistant-1.5.0-patch-1.

1. Use `oc get wa` to see your instance name:

```
oc get wa
NAME                    VERSION   READY   READYREASON   UPDATING   UPDATINGREASON   DEPLOYED   VERIFIED   AGE
watson-assistant---wa-qa   1.5.0     True    Stable        False      Stable           18/18      18/18      11h
```

2. Export your instance name as a variable that you can use in each step, for example:

```
export INSTANCENAME=watson-assistant---wa-qa
```

3. Change the `updateStrategy` in both Redis statefulsets to type `RollingUpdate`:

```
oc patch statefulset c-$INSTANCENAME-redis-m -p '{"spec":{"updateStrategy":{"type":"RollingUpdate"}}}'
oc patch statefulset c-$INSTANCENAME-redis-s -p '{"spec":{"updateStrategy":{"type":"RollingUpdate"}}}'
```

4. Update the Redis statefulsets with the resized cpu and memory values:

   - Member CPU

```
oc patch statefulset c-$INSTANCENAME-redis-m --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/requests/cpu", "value":"50m"}]'
```

   - Member memory

```
oc patch statefulset c-$INSTANCENAME-redis-m --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/0/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/requests/memory", "value":"256Mi"}]'
```

   - Sentinel CPU

```
oc patch statefulset c-$INSTANCENAME-redis-s --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/requests/cpu", "value":"50m"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/requests/cpu", "value":"50m"}]'
```

   - Sentinel memory

```
oc patch statefulset c-$INSTANCENAME-redis-s --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/limits/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/0/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/1/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/2/resources/requests/memory", "value":"256Mi"},{"op": "replace", "path": "/spec/template/spec/containers/3/resources/requests/memory", "value":"256Mi"}]'
```

5. Confirm that the Redis member and sentinel pods have the new memory and cpu values, for example:

```
oc describe pod c-$INSTANCENAME-redis-m-0 |grep cpu
oc describe pod c-$INSTANCENAME-redis-m-0 |grep memory
oc describe pod c-$INSTANCENAME-redis-s-0 |grep cpu
oc describe pod c-$INSTANCENAME-redis-s-0 |grep memory
```

6. The results should look like these examples:

```
oc describe sts c-$INSTANCENAME-redis-m |grep cpu
                {"m":{"db":{"limits":{"cpu":"4","memory":"256Mi"},"requests":{"cpu":"25m","memory":"256Mi"}},"mgmt":{"limits":{"cpu":"2","memory":"100Mi"}...
    cpu:     4
    cpu:     50m
    cpu:     2
    cpu:     50m
    cpu:     2
    cpu:      50m
    cpu:     2
    cpu:     50m
```

```
oc describe sts c-$INSTANCENAME-redis-m |grep memory
                {"m":{"db":{"limits":{"cpu":"4","memory":"256Mi"},"requests":{"cpu":"25m","memory":"256Mi"}},"mgmt":{"limits":{"cpu":"2","memory":"100Mi"}...
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
    memory:   256Mi
    memory:  256Mi
    memory:  256Mi
```

```
oc describe pod c-$INSTANCENAME-redis-s-0 |grep cpu
            {"m":{"db":{"limits":{"cpu":"4","memory":"256Mi"},"requests":{"cpu":"25m","memory":"256Mi"}},"mgmt":{"limits":{"cpu":"2","memory":"100Mi"}...
    cpu:     2
    cpu:     50m
    cpu:     2
    cpu:     50m
    cpu:     2
    cpu:      50m
    cpu:     2
    cpu:     50m
```

```
oc describe pod c-$INSTANCENAME-redis-s-0 |grep memory
            {"m":{"db":{"limits":{"cpu":"4","memory":"256Mi"},"requests":{"cpu":"25m","memory":"256Mi"}},"mgmt":{"limits":{"cpu":"2","memory":"100Mi"}...
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
    memory:   256Mi
    memory:  256Mi
    memory:  256Mi
    memory:  256Mi
```

7. Change the `updateStrategy` in both Redis statefulsets back to type `OnDelete`:

```
oc patch statefulset c-$INSTANCENAME-redis-m -p '{"spec":{"updateStrategy":{"type":"OnDelete"}}}'
oc patch statefulset c-$INSTANCENAME-redis-s -p '{"spec":{"updateStrategy":{"type":"OnDelete"}}}'
```

Delete the pdb (poddisruptionbudgets) when changing the deployment from medium to small

Whenever the deployment size is changed from medium to small. A manual step is required to delete the `poddisruptionbudgets` that are created for medium instances.

Run the following command, replacing `<instance-name>` with the name of your CR instance and replacing `<namespace-name>` with the name of the namespace where the instance resides.

```
oc delete pdb  -l icpdsupport/addOnId=assistant,component!=etcd,ibmevents.ibm.com/kind!=Kafka,app.kubernetes.io/instance=<instance-name> -n <namespace-name>
```

Using the Cloud Object Storage importer to migrate chat logs

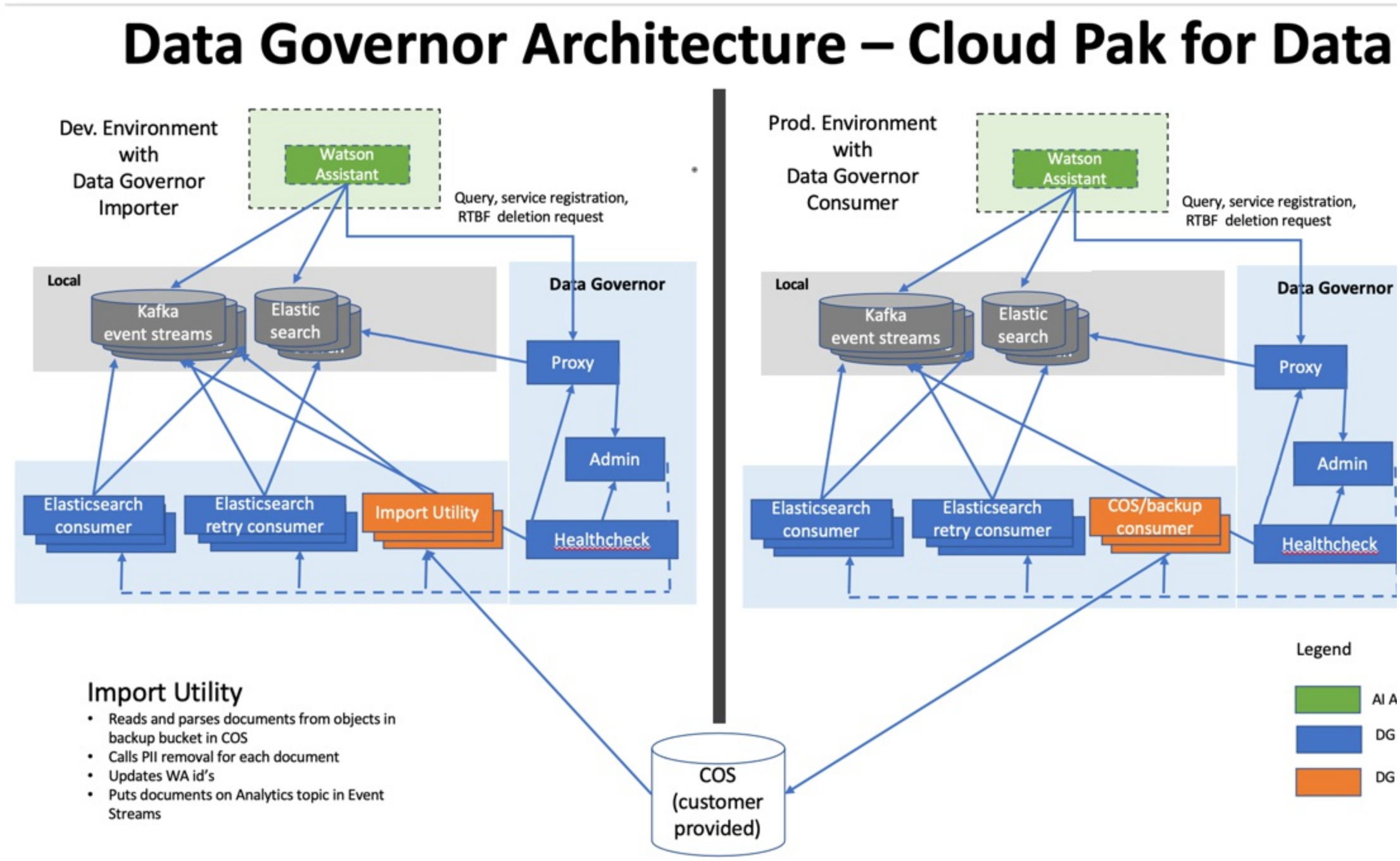
IBM Cloud Pak for Data

You can use the Cloud Object Storage importer service to migrate your chat logs from one installation to another.

This diagram shows two clusters:

Cluster	Description
Cloud Object Storage consumer	Red Hat OpenShift cluster with the installation used to export chat log data to Cloud Object Storage
Cloud Object Storage importer	Red Hat OpenShift cluster with the installation that imports chat log data from the copied bucket from the Cloud Object Storage consumer cluster

Diagram clusters



{caption="Architecture Diagram" caption-side="bottom"}

Configuring the data governor Cloud Object Storage consumer service

1. Use the script to create a new file on the Cloud Object Storage consumer cluster.

```
COS_ACCESS_KEY=`echo -n "<enter_COS_key_here>"|base64`
COS_SECRET_ACCESS_KEY=`echo -n "<enter_COS_secret_access_key_here>"|base64`
COS_ENDPOINT="<enter_COS_endpoint here>"
#For e.g. COS_ENDPOINT="https://s3.us.cloud-object-storage.appdomain.cloud"

cat <<EOF | oc apply -f -

apiVersion: v1
kind: Secret
metadata:
```

```
  name: wa-data-governor-cos-credentials
  namespace: cpd
data:
  COS_ACCESS_KEY: ${COS_ACCESS_KEY}
  COS_SECRET_KEY: ${COS_SECRET_ACCESS_KEY}
type: Opaque

EOF

cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: wa-data-governor
  namespace: cpd
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistant
  name: wa
  patch:
    data-governor:
      dataexhaust:
        spec:
          consumers:
            cosConsumer: true
          cos:
            endpoint: ${COS_ENDPOINT}
            secret:
              accessKey: COS_ACCESS_KEY
              name: wa-data-governor-cos-credentials
              secretKey: COS_SECRET_KEY
  patchType: patchStrategicMerge
EOF
```

2. In the script, replace these placeholders with the actual values:
   - `<enter_COS_key_here>`
   - `<enter_COS_secret_access_key_here>`
   - `<enter_COS_endpoint here>`
3. Run the script.
4. Wait 15 to 20 minutes and then check if the data governor pods are stable.
5. When the pods are stable, log in to your instance.
6. Select the correct assistant, then click the `Preview` icon on the left vertical bar and generate chat logs.
7. Check if the Cloud Object Storage objects are getting created in the `data-exhaust-backup-wa-data-governor-e58d5f4a1344` Cloud Object Storage bucket.

Configuring the data governor Cloud Object Storage importer service

Create a file on the Cloud Object Storage Importer cluster and copy the following script into it. Replace the placeholders `<enter_COS_key_here>`, `<enter_COS_secret_access_key_here>`, and `<enter_COS_endpoint here>` with the actual values and save the file. Then, run the script. Wait 15-20 minutes and check if the data governor pods are stable. When the pods are stable, check if the importer cron is running as scheduled. When the `importer` pod completes, check if chat logs data were imported in the correct assistant chat logs by going to Assistant UI and checking the `Analyze` screen for imported data.

1. Use the script to create a new file on the Cloud Object Storage importer cluster.

```
COS_ACCESS_KEY=`echo -n "<enter_COS_key_here>"|base64`
COS_SECRET_ACCESS_KEY=`echo -n "<enter_COS_secret_access_key_here>"|base64`

cat <<EOF | oc apply -f -

apiVersion: v1
kind: Secret
metadata:
  name: wa-data-governor-cos-credentials
  namespace: cpd
```

```
  data:
    COS_ACCESS_KEY: ${COS_ACCESS_KEY}
    COS_SECRET_KEY: ${COS_SECRET_ACCESS_KEY}
  type: Opaque

  EOF
```

```
COS_ENDPOINT="<enter_COS_endpoint here>"  # For e.g. COS_ENDPOINT="https://s3.us.cloud-object-storage.appdomain.cloud"
# See "How to generate a JSON mapping defintion" below for instructions on how to create the next line
JSON_ID_MAPPING='{"instances":[{"source_id":"<instance_id_consumer1>","target_id":"<instance_id_importer1>","assistants":[{"source_id":"
<draft_or_live_environment_id_consumer1>","target_id":"<draft_or_live_environment_id_importer1>"}],"workspaces":[{"source_id":"
<action_skill_id_consumer1>","target_id":"<action_skill_id_importer1>"}]}]}'

cat <<EOF | oc apply -f -
apiVersion: assistant.watson.ibm.com/v1
kind: TemporaryPatch
metadata:
  name: wa-data-governor
spec:
  apiVersion: assistant.watson.ibm.com/v1
  kind: WatsonAssistant
  name: wa
  patchType: patchStrategicMerge
  patch:
    data-governor:
      dataexhaust:
        spec:
          cos:
            endpoint: ${COS_ENDPOINT}
            secret:
              accessKey: COS_ACCESS_KEY
              name: wa-data-governor-cos-credentials
              secretKey: COS_SECRET_KEY
          cosImporter:
            fromBucket: data-exhaust-backup-wa-data-governor-e58d5f4a1344
            clearObjectTags: false
            filter:
# Modify prefix value.
              prefix: "20230322"
            fromTenant: "*"
# Modify schedule to run importer cron less frequently
            schedule: '* * * * *'
# See "How to generate a JSON mapping defintion" below for instructions on how to create the next line
            schema: ${JSON_ID_MAPPING}
            workers: 3
```

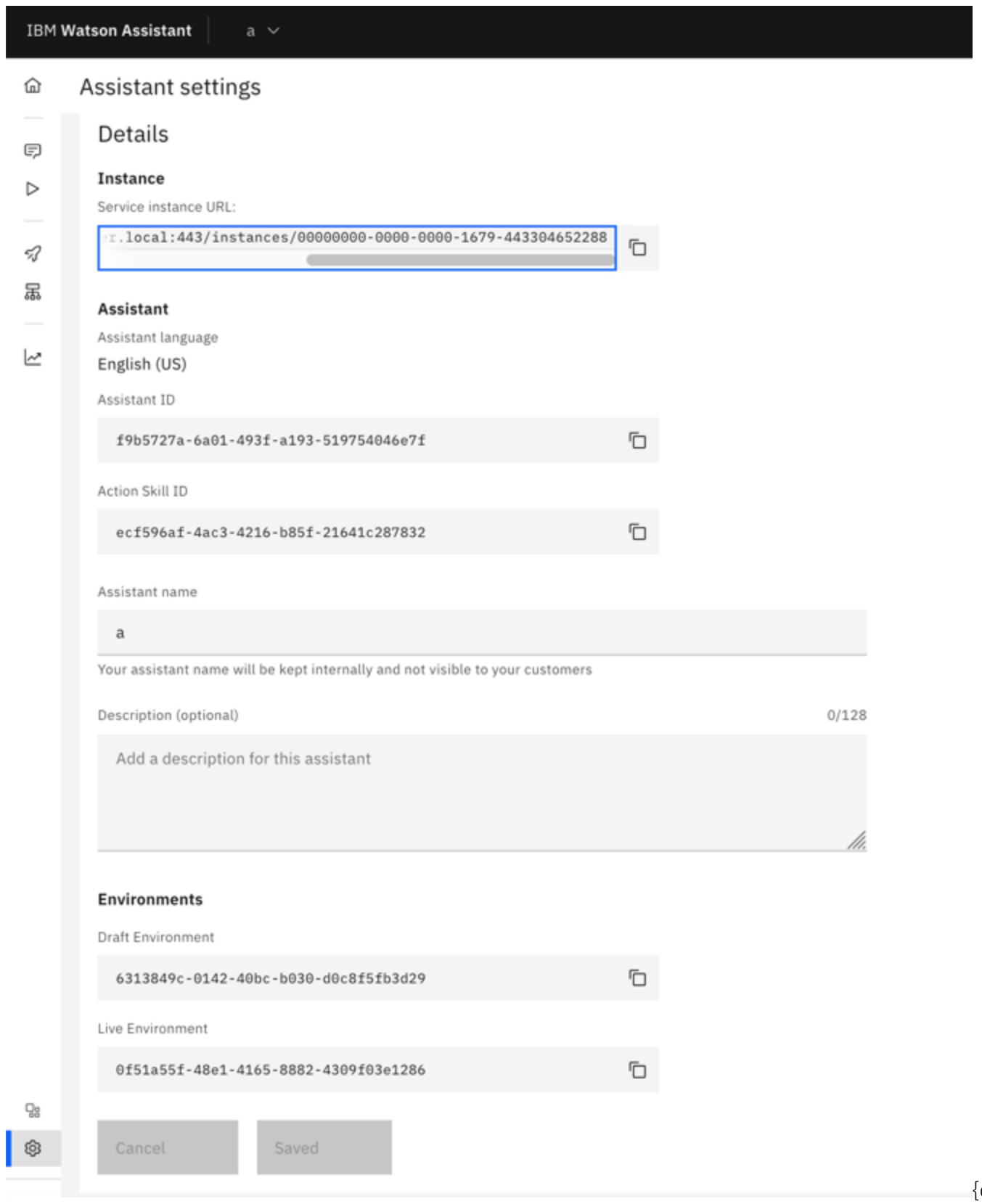
2. In the script, replace these placeholders with the actual values:
   - `<enter_COS_key_here>`
   - `<enter_COS_secret_access_key_here>`
   - `<enter_COS_endpoint here>`
3. Run the script.
4. Wait 15 to 20 minutes and then check if the data governor pods are stable.
5. When the pods are stable, check that the importer cron is running as scheduled.
6. When the `importer` pod completes, check that chat logs were imported in the correct assistant by opening your assistant and checking the `Analyze` screen for imported data.

Parameter	Description
fromBucket	Cloud Object Storage bucket with the data to be imported
schedule	Schedule for Cloud Object Storage cron job to run

clearObjectTags	Set to true to allow the importer to rerun on already processed Cloud Object Storage objects
workers	Number of worker threads. Increasing the number requires more job resources.
filter	Provides control over which objects are imported from the Cloud Object Storage bucket
prefix	Allows for objects from a specific date to be imported
documentTimestampWindow	Allows for objects during a specific time range to be imported
cos	Cloud Object Storage credentials
schema	Transformation schema, which allows the importer to modify fields to allow analytic data from pprd instance to be viewed in train. For more information, see Generating a JSON ID mapping definition for the data governor importer service.
fromTenant	Allows import on specific filter tenant data

Parameters

Tip: `instance_id_consumer1`, `draft_or_live_environment_id_consumer1`, and `action_skill_id_consumer1` values can be found on the Assistant settings page in on the Cloud Object Storage consumer cluster.



{caption="Assistant settings" caption-side="bottom"}

For example:

- `instance_id_consumer1` can be extracted from the `Service Instance URL`. For example, `00000000-0000-0000-1679-443304652288`.
- `draft_or_live_environment_id_consumer1` values are the `Draft environment` or `Live Environment` IDs.
- `action_skill_id_consumer1` value is the `Action Skill ID`.

Similarly, you can extract Cloud Object Storage importer instances, assistants, and workspaces target_id values from assistant settings for the Cloud Object Storage importer cluster.

Generating a JSON ID mapping definition for the data governor importer service

You can use the Analytics reports in your development environment to gain insight into your production chat logs.

You need to move your production chat logs data to your development environment, and then transform the production chat logs data to be used by the Analytics reports in your development assistants.

For the moved chat logs to be used by the Analytics reports in a different instance and assistant in your development environment, the various IDs of the chat log Elastic documents need to be transformed from the source to the target.

Use this code for the ID mapping definition.

- This mapping definition supports transforming the chat logs for multiple instances and assistants.
- The mapping is hierarchical, so each instance level mapping also contains multiple assistant-level mappings for such instance.
- Each mapping contains two ID fields: `source_id` and `target_id`. You need to supply data for each mapping at the instance and assistant levels.

You can look up the ID data in applicable instances and assistants.

```
{
 "instances": [
  {
   "source_id": "<string>",
   "target_id": "<string>",
   "assistants": [
    {
     "source_id": "<string>",
     "target_id": "<string>"
    }
   ],
   "workspaces": [
    {
     "source_id": "<string>",
     "target_id": "<string>"
    }
   ]
  }
 ]
}
```

Usage examples

Map assistants from one source instance to assistants in multiple target instances.

The following example maps 2 assistants in 1 source instance to 2 assistants in 2 target instances.

```
{
 "instances": [
  {
   "source_id": "07079e0a-a8cd-4dc5-b3a4-7879d7cf0a8f",
   "target_id": "90876f0b-c98d-fcc9-c4b5-0291a3bd4f9a",
   "assistants": [
    {
     "source_id": "081513d5-9858-4eb9-9012-5ce0af8182a6",
     "target_id": "f4e5f6d5-3903-486f-8563-416446d8c8a0"
    }
   ]
  },
  {
   "source_id": "07079e0a-a8cd-4dc5-b3a4-7879d7cf0a8f",
   "target_id": "79640a7b-45bc-7eea-b4f1-98b55fca6b1c",
```

```
   "assistants": [
    {
     "source_id": "9645f5d1-f432-4abd-a589-12bc4af5b133",
     "target_id": "a4bb579e-e3b5-34bc-afe5-9afc490e8bac"
    }
   ]
  }
 ]
}
```

Glossary

Term	Definition	Applies to
Action	Actions represent the tasks or questions that your assistant can help customers with. Each action has a beginning and an end, making up a conversation between the assistant and a customer. Learn more.	Actions
Ask clarifying question	A feature that enables the assistant to ask customers to clarify (disambiguate) their meaning when the assistant isn't sure what a user wants to do next. Learn more.	Actions
Assistant	Container for your actions, channels, and integrations. You add actions and at least one channel to an assistant, and then deploy the assistant when you are ready to start helping your customers. Learn more.	Both
Change conversation topic	A feature that gives the user the power to direct the conversation. It allows digressions and prevents customers from getting stuck in a dialog thread. They can switch topics whenever they choose. Learn more.	Actions
Channel	The location where your assistant interacts with your users, for example, over the phone, on a website, or in Slack. At least one channel is required for every assistant. Learn more.	Both
Completion	Measures how often within a time period that a user reaches the end step of an action. Learn more.	Actions
Content	The conversation logic and words that are used to respond to your customer. Content is required for every assistant. Learn more.	Both
Content catalog	A set of prebuilt intents that are categorized by subject, such as customer care. You can add these intents to your dialog and start using them immediately. Or you can edit them to complement other intents that you create. Learn more.	Dialog
Context variable	A variable that you can use to collect information during a conversation, and reference it later in the same conversation. For example, you might want to ask for the customer's name and then address the person by name later on. A context variable is used by dialog. Learn more.	Dialog
Dialog	A component where you can build the conversation that your assistant has with your customers. For each defined intent, you can write the response your assistant returns. Learn more.	Dialog
Digression	A dialog feature that gives the user the power to direct the conversation. It prevents customers from getting stuck in a dialog thread; they can switch topics whenever they choose. Learn more.	Dialog
Disambiguation	A dialog feature that enables the assistant to ask customers to clarify their meaning when the assistant isn't sure what a user wants to do next. Learn more.	Dialog
Entity	Information in the user input that is related to the user's purpose. An intent represents the task that a user wants to do. An entity represents the object of that action. Learn more.	Dialog
Environment	You can group your work in separate containers that are called *environments*. Each environment contains its own content, channels, and extensions. Environments also have their own IDs, URLs, and service credentials that can be referenced by external services. Each new assistant comes with two environments: the draft environment and the live environment. Your customers interact with assistants on the live environment and cannot interact with assistants on the draft environment. The separation of these two environments allows you to build and iterate on your content separately from what your customers see. You do not want customers to stumble upon an incomplete action that leads them to a dead end. For Enterprise plans, you can add up to three environments as a staging area to test your assistant before deployment. You can build content in the draft environment and test versions of your content in the extra environments. Learn more.	Both
Escalation	If your assistant is integrated with one of the supported service desk systems, you can build in logic that transfers, or escalates, the conversation to a person when necessary. Learn more.	Both
Extension	A custom extension is an integration with an external service. By calling an extension from an action, your assistant can send requests to the external service and receive response data it can use in the conversation. Learn more.	Actions

Incompletion	Reasons why an action is not completed by a user, including `escalated to agent`, `started a new action`, `stuck on a step`, or `abandoned or ongoing`. Learn more	Actions
Integrations	Add-ons to the end experience that help solve specific user problems, for example, connecting to a human agent or searching existing help content. Integrations are not required for an assistant, but they are recommended. Learn more.	Both
Intent	The goal that is expressed in the user input, such as answering a question or processing a bill payment. Learn more.	Dialog
Message	A single turn within a conversation that includes a single call to the `/message` API endpoint and its corresponding response.	Both
Monthly active user (MAU)	A single unique user who interacts with an assistant one or many times in a month. Learn more.	Both
Preview	Embeds your assistant in a chat window that is displayed on an IBM-branded web page. From the preview, you can test how a conversation flows through your assistant, from end to end. Learn more.	Actions
Recognition	Measurement of how many requests are being recognized by the assistant and routed into starting an action. Learn more.	Actions
Response	To create your assistant's response in an action step, you use the *Assistant says* section. This represents the text or speech that the assistant delivers to a user at a particular step. Depending on the step, you can add a complete answer to a user's question or ask a follow-up question. Learn more.	Actions
Response	Logic that is defined in the *Assistant responds* section of a dialog node that determines how the assistant responds to the user. When the node's condition evaluates to true, the response is processed. The response can consist of an answer, a follow-up question, a webhook that sends a programmatic request to an external service, or slots that represent pieces of information that you need the user to provide before the assistant can help. The dialog node response is equivalent to a Then statement in If-Then-Else programming logic.	Dialog
Slots	A special set of fields that you can add to a dialog node that enable the assistant to collect necessary pieces of information from the customer. For example, the assistant can require a customer to provide valid date and location details before it gets weather forecast information on the customer's behalf. Learn more.	Dialog
Step	A step that you add to an action represents a single interaction or exchange of information with a customer, a turn in the conversation. Learn more.	Actions
System entity	Prebuilt entities that recognize references to common things like dates and numbers. You can add these to your dialog and start using them immediately. Learn more.	Dialog
Subaction	An action that is called from another action is referred to as a *subaction*. Learn more.	Actions
Try it out	A chat window that you use to test as you build. For example, from the dialog skill's "Try it out" pane, you can mimic the behavior of a customer and enter a query to see how the assistant responds. You can test only the current skill; you cannot test your assistant and all attached skills from end to end. Learn more.	Dialog
Training	To set up an instance with components that enable the system to function in a particular domain (for example: corpus content, training data that generates machine learning models, programmatic algorithms, annotators, or other ground truth components) and then making improvements and updates to these components based on accuracy analysis.	Both
Variable	A variable is data that a customer shares with the assistant, which is collected and saved so it can be referenced later. In actions, you can collect *action* and *session* variables. Learn more.	Both
Web chat	A channel that you can use to embed your assistant in your company website. Learn more.	Both
Webhook	A mechanism for calling out to an external program during a conversation. For example, your assistant can call an external service to translate a string from English to French and back again in the course of the conversation. Learn more.	Both

Response types reference

You can use the JSON editor to specify responses of many different types. You can use JSON editor to specify the responses to the customer query. By adding the JSON scripts in the JSON editor, your assistant uses the response format mentioned in the JSON script.

For more information, see Defining responses with the JSON editor.

When the variables in the action and message API differ at run time, the format of response types in the action and message API also differs. The following examples show the differences in response type format when you use message API and the JSON editor.

If the **text** response from the message API has the following format:

```
{ "response_type": "text", "text": "Hello world" }
```

Then, the assistant displays the actual text message, `Hello world`, in a single step.

If the **text** response from the JSON action editor has the following format:

```
$ {
  "generic": [
   {
     "response_type": "text",
    "values": [
     {
       "text_expression": {
         "concat": [
           {
             "scalar": "Hi, "
           },
           {
             "variable": "step_472"
           },
           {
             "scalar": ". How can I help you?"
           }
         ]
       }
     }
    ],
    "selection_policy": "sequential"
   }
  ]
}
```

Then, the assistant combines actual value of `variable` with other items in the `values` array and displays the response. For example, if `step_472` takes the value "Bob", then the assistant displays `Hi, Bob. How can I help you?`.

Viewing the response type at the runtime

You can refer to the API documentation for watsonx Assistant to view the details of response types and the APIs.

For example, to view the runtime response type, do the following:

1. In the **Response** section, click `MessageOutput` in **output** to see the **generic** section.
2. In the `generic` section, click `RuntimeResponseGeneric[]`.
3. Select an option in the `One of` dropdown.

Note: To view more details about the selected option, click `One of`.

Response

StatefulMessageResponse A response from the watsonx Assistant service.

output
Always included *

Assistant output to be rendered or processed by the client.

MessageOutput

generic

Output intended for any channel. It is the responsibility of the client application to implement the supported response types.

RuntimeResponseGeneric[]

One of

RuntimeResponseGenericRuntimeResponseTypeText

The following response types are supported in the JSON editor.

audio

Plays an audio clip that is specified by a URL.

Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✔	✔	✔	✔	✔	✔

- Some channel integrations do not display audio titles or descriptions.

Fields

Name	Type	Description	Required?
response_type	string	`audio`	Y
source	string	The `https:` URL of the audio clip. The URL can specify either an audio file or an audio clip on a supported hosting service.	Y
title	string	The title to show before the audio player.	N
description	string	The text of the description that accompanies the audio player.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the audio player cannot be seen.	N
channel_options.voice_telephony.loop	string	Whether the audio clip repeats indefinitely (phone integration only).	N

The URL specified by the `source` property can be one of the following types:

- The URL of an audio file in any standard format such as MP3 or WAV. In the web chat, the linked audio clip renders as an embedded audio player.
- The URL of an audio clip on a supported streaming service. In the web chat, the linked audio clip renders by using the embeddable player for the hosting service.

Specify the URL that you use to access the audio file in your browser (for example, `https://soundcloud.com/ibmresearchfallen-star-amped`). The web chat automatically converts the URL to an embeddable form.

You can embed audio hosted on a supported service:

- SoundCloud
- Mixcloud

Note: For the phone integration, the URL must specify an audio file that is single-channel (mono) and PCM-encoded, and is sampled at 8,000 Hz with 16 bits per sample.

Example

This example plays an audio clip with a title and descriptive text.

```
{
  "generic":[
    {
      "response_type": "audio",
      "source": "https://example.com/audio/example-file.mp3",
      "title": "Example audio file",
      "description": "An example audio clip returned as part of a multimedia response."
    }
  ]
}
```

button

Show interactive buttons that help the users to complete their tasks.

Integration channel support

WebChat
✔

Fields

Name	Type	Description	Required?
response_type	string	`button`	Y
label	string	The button label	Y
button_type	string	The type of button. For example, `post_back`, `custom_event`, `show_panel`, and `url`.	Y
kind	string	The kind of button. For example, `primary`, `secondary`, `tertiary`, `danger`, and `link`. The default value is `primary`.	N
image_url	string	The url of an image to render as a button.	N
alt_text	string	The alternate text to label the image for the accessibility purposes.	N

`post_back` button type

The `post_back` button sends a response to the assistant when the user clicks the button. You can use both the `value` and `label` properties to set up the response to send.

Fields

Name	Type	Description	Required?

value	object	Defines the response that WebChat sends to the watsonx Assistant service when the user selects an option. **Note:** If you do not define `value.input.text`, then the WebChat sends the value of `label` to the assistant.	N

Example

The following example shows the JSON configuration for a button to send an authored text input to the assistant when the user clicks the button.

```
{
  "response_type": "button",
  "button_type": "post_back",
  "label": "Send message",
  "value": {
    "input": {
      "text": "[Message to send]"
    }
  }
}
```

`custom_event` button type

When the user clicks the `custom_event` button, a custom event, which you configured along with the user-defined data, is triggered. You must create the event to achieve the desired behavior by using custom codes. In web chat, you can use the `messageItemCustom` event to apply the desired behavior when the user clicks the button.

Fields

Name	Type	Description	Required?
custom_event_name	string	The name of the custom event that is triggered when the user clicks the button. In web chat, the messageItemCustom event is triggered when the user clicks the `custom_event` button.	Y
user_defined	object	The user-defined data attached to the custom event.	N

Example

In the following example, you use the JSON script to trigger an alert when the button is clicked.

```
{
  "response_type": "button",
  "button_type": "custom_event",
  "label": "Alert",
  "kind": "danger",
  "custom_event_name": "trigger_alert",
  "user_defined": {
    "message": "[Alert message]"
  }
}
```

`show_panel` button type

When a user clicks the `show_panel` button, the assistant opens a panel that the users can use to get additional information or complete a task.

Fields

Name	Type	Description	Required?
panel	string	An object that defines the content of the panel.	Y
panel.title	string	The title of the panel.	N

panel.show_animations	boolean	The object to enable or disable the animations in the panel when the user opens or closes it. The default value is `true`	Y
panel.body[]	list	A list of response types to create rich visual content. Maximum 10 response types are allowed in the list. Supported response types: `text`, `image`, `video`, `audio`, `iframe`, `grid`, `card` and `user_defined`. **Note:** A card response type in a panel does not support buttons.	Y
panel.footer[]	list	A list of `button` response types. Maximum 5 buttons are allowed in the list. **Note:** The button type `show_panel` is not supported in this list.	N

Example

In the following example, you use JSON script to create a button that opens a panel for product details:

```
{
  "response_type": "button",
  "button_type": "show_panel",
  "label": "See details",
  "kind": "secondary",
  "panel": {
    "title": "[Product name]",
    "show_animations": true,
    "body": [
      {
        "response_type": "image",
        "source": "https://example.com/image.jpg"
      },
      {
        "response_type": "text",
        "text": "[Product details]"
      }
    ]
  }
}
```

`url` button type

When a user clicks the `url` button, the user goes to the url field to add the destination url.

Fields

Name	Type	Description	Required?
url	string	The destination url of the button.	Y
target	string	The location to open the url in the browser. For example, `_blank` or `_self`. `_blank` opens the url in a new tab. `_self` opens the url in the same tab. The default value is `_blank`.	N

Example

The example presents a button that takes the user to ibm.com when the button is clicked.

```
{
  "response_type": "button",
  "button_type": "url",
  "label": "Visit ibm.com",
  "url": "https://www.ibm.com"
}
```

card

Visual content to improve the information experience of users by using `card`.

Integration channel support

WebChat
✔

Fields

Name	Type	Description	Required?
response_type	string	`card`	Y
body[]	list	A list of response types to create rich content. A maximum of 10 response types are allowed in the list. Supported response types: `text`, `image`, `video`, `audio`, `iframe`, `grid`, and `user_defined`.	Y
footer[]	list	A list of only `button` response types. A maximum of 5 buttons are allowed in the list.	N

A `card` can be rendered in a panel, but it is not allowed to have buttons.

Example

The following example shows the basic structure for building a `card` response type:

```
{
  "response_type": "card",
  "body": [
    {
      "response_type": "text",
      "text": "# Heading"
    },
    {
      "response_type": "text",
      "text": "body"
    }
  ]
}
```

carousel

A `carousel` to present cards with rich content. If there is only one card in the carousel, the web chat integration will just render the card instead of the card in a carousel.

Integration channel support

WebChat
✔

Fields

Name	Type	Description	Required?
response_type	string	`carousel`	Y
items[]	list	A list of `card` response types. A maximum of 5 cards are allowed in the list.	Y

Example

The following example shows the basic structure for building a `carousel` response type:

```
{
  "response_type": "carousel",
  "items": [
    {
      "response_type": "card",
      "body": [
        {
          "response_type": "text",
          "text": "# Heading"
        },
        {
          "response_type": "text",
          "text": "body"
        }
      ]
    },
    {
      "response_type": "card",
      "body": [
        {
          "response_type": "text",
          "text": "# Heading"
        },
        {
          "response_type": "text",
          "text": "body"
        }
      ]
    }
  ]
}
```

channel_transfer

Transfers the conversation to a different channel integration. Currently, the web chat integration is the only supported target of a channel transfer.

Integration channel support

Phone	SMS	Slack	Facebook	WhatsApp
✔	✔	✔	✔	✔

- The indicated channel integrations support *initiating* a channel transfer (currently, the web chat integration is the only supported transfer target).
- Initiating a channel transfer from the phone integration requires that the SMS integration is also configured.

Fields

Name	Type	Description	Required?
response_type	string	`channel_transfer`	Y
message_to_user	string	A message to display to the user before the link for initiating the transfer.	Y
transfer_info	object	Information used by an integration to transfer the conversation to a different channel.	Y
transfer_info.target.chat	string	URL for the website that hosts the web chat to which the conversation is transferred.	Y

Example

This example requests a transfer from WhatsApp to the web chat. In addition to the `channel_transfer` response, the output also includes a `text` response to be displayed by the web chat integration after the transfer. The use of the `channels` array ensures that the `channel_transfer` response is handled only by the WhatsApp integration (before the transfer), and the `connect_to_agent` response only by the web chat integration (after the transfer).

```
{
  "generic": [
    {
      "response_type": "channel_transfer",
      "channels": [
        {
          "channel": "whatsapp"
        }
      ],
      "message_to_user": "Click the link to connect with an agent using our website.",
      "transfer_info": {
        "target": {
          "chat": {
            "url": "https://example.com/webchat"
          }
        }
      }
    },
    {
      "response_type": "connect_to_agent",
      "channels": [
        {
          "channel": "chat"
        }
      ],
      "message_to_human_agent": "User asked to speak to an agent.",
      "agent_available": {
        "message": "Please wait while I connect you to an agent."
      },
      "agent_unavailable": {
        "message": "I'm sorry, but no agents are online at the moment. Please try again later."
      },
      "transfer_info": {
        "target": {
          "zendesk": {
            "department": "Payments department"
          }
        }
      }
    }
  ]
}
```


connect_to_agent

Transfers the conversation to a live agent for help. Service desk support must be configured for the channel integration.

Integration channel support

Web chat	Phone	WhatsApp
✔	✔	✔

- For information about adding service desk support to the web chat integration, see Adding contact center support.
- For information about adding service desk support to the phone integration, see Configuring backup call center support.

Fields

Name	Type	Description	Required?
response_type	string	`connect_to_agent`	Y
message_to_human_agent	string	A message to display to the live agent to whom the conversation is being transferred.	Y
agent_available	string	A message to display to the user when agents are available.	Y

agent_unavailable	string	A message to display to the user when no agents are available.	Y
transfer_info	object	Information that is used by the web chat service desk integrations for routing the transfer.	N
transfer_info.target.zendesk.department	string	A valid department from your Zendesk account.	N
transfer_info.target.salesforce.button_id	string	A valid button ID from your Salesforce deployment.	N

Example

This example requests a transfer to a live agent and specifies messages to be displayed both to the user and to the agent at the time of transfer.

```
{
  "generic": [
    {
      "response_type": "connect_to_agent",
      "message_to_human_agent": "User asked to speak to an agent.",
      "agent_available": {
        "message": "Please wait while I connect you to an agent."
      },
      "agent_unavailable": {
        "message": "I'm sorry, but no agents are online at the moment. Please try again later."
      }
    }
  ]
}
```

date

Use an interactive date picker that a customer can use to specify a date value.

Integration channel support

Web chat
✔

- In the web chat, the customer can specify a date value either by clicking the interactive date picker or typing a date value in the input field.

Fields

Name	Type	Description	Required?
response_type	string	`date`	Y

Example

This example sends a text response that asks the user to specify a date, and then shows an interactive date picker.

```
{
  "generic": [
    {
      "response_type": "text",
      "text": "What day will you be checking in?"
    },
    {
      "response_type": "date"
    }
  ]
}
```

dtmf

Sends commands to the phone integration to control input or output with dual-tone multi-frequency (DTMF) signals. (DTMF is a protocol that transmits tones, which are generated when a user presses keys on a push-button phone.)

Integration channel support

Phone
✔

Fields

Name	Type	Description	Required?
response_type	string	`dtmf`	Y
command_info	object	Information specifying the DTMF command to send to the phone integration.	Y
command_info.type	string	The DTMF command to send (`collect`, `disable_barge_in`, `enable_barge_in`, or `send`).	Y
command_info.parameters	object	See Handling phone interactions.	N

The `command_info.type` field can specify any of the following supported commands:

- `collect`: Collects DTMF keypad input.
- `disable_barge_in`: Disables DTMF barge-in so that playback from the phone integration is not interrupted when the customer presses a key.
- `enable_barge_in`: Enables DTMF barge-in so that the customer can interrupt playback from the phone integration by pressing a key.
- `send`: Sends DTMF signals.

For detailed information about how to use each of these commands, see Handling phone interactions.

Example

This example shows the `dtmf` response type with the `collect` command, used to collect DTMF input. For more information, see Handling phone interactions.

```
{
  "generic": [
    {
      "response_type": "dtmf",
      "command_info": {
        "type": "collect",
        "parameters": {
          "termination_key": "#",
          "count": 16,
          "ignore_speech": true
        }
      },
      "channels": [
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}
```

end_session

Sends a command to the channel that ends the session. This response type instructs the phone integration to hang up the call.

Integration channel support

Phone	SMS

✔

- The SMS integration supports ending a session by using the `terminateSession` action command.

Fields

Name	Type	Description	Required?
response_type	string	`end_session`	Y

For the phone integration, you can use the `channel_options` object to include custom headers with the SIP `BYE` request that is generated. For more information, see End the call.

Example

This example uses the `end_session` response type to end a conversation.

```
{
  "generic": [
    {
      "response_type": "end_session"
    }
  ]
}
```

grid

Gives you the flexibility to create the layout you need to present content that conveys the type of information you want users to consume.

Integration channel support

WebChat
✔

Fields

Name	Type	Description	Required?
response_type	string	`grid`	Y
horizontal_alignment	string	The horizontal alignment for all items in the grid (`left`, `center`, or `right`). The default value is `left`.	N
vertical_alignment	string	The vertical alignment for all items in the grid (`top`, `center`, or `bottom`). The default values is `top`.	N
columns[]	list	The list of columns. A maximum of 5 columns are allowed in the list. Each column is separated by 8px of space.	N
columns[].width	string	The width of the column. You can set the value of width by using number (for example, `1`) or pixel (for example, `48 px`). The number value of a column width is calculated based on the total width of the row and the width of other columns in the row. For example, if the width of the first column is `1` and the width of the second column is 2, then the first column and the second column takes one-third and two-thirds of the total width of the row respectively. By default, the number value of width is `1`.	Y

rows[]	list	The list of rows. Maximum 5 rows are allowed in the list. Each row is separated by 8px of space.	Y
rows[].cells[]	list	The list of cells in a row. Each cell is a column in a row (for example, cell 1 is column 1 in a row). The width of the cell is equal to width of the column.	Y
rows[].cells[].items[]	list	A list of response-type items. Each item is separated by 8px of space. Maximum 5 response-type items are allowed in the list. Supported response-type items are `text`, `image`, `vdeo`, `audio`, `iframe`, `grid`, and `user_defined`. You can set up a `grid` only within a `grid` and below the first level. A `grid` in a cell cannot contain a grid response type.	Y
rows[].cells[].horizontal_alignment	string	The horizontal alignment for items in the cell (`left`, `center`, or `right`). The default value is `left`.	N
rows[].cells[].vertical_alignment	string	The vertical alignment for items in the cell (`top`, `center`, or `bottom`). The default values is `top`.	N

Example

The following example shows the basic structure for building a `grid` response type:

```
{
  "response_type": "grid",
  "columns": [
    {
      "width": "1"
    },
    {
      "width": "1"
    }
  ],
  "rows": [
    {
      "cells": [
        {
          "items": [
            {
              "response_type": "text",
              "text": "row 1 cell 1"
            }
          ]
        },
        {
          "items": [
            {
              "response_type": "text",
              "text": "row 1 cell 2"
            }
          ]
        }
      ]
    },
    {
      "cells": [
        {
          "items": [
            {
              "response_type": "text",
              "text": "row 2 cell 1"
            }
          ]
```

```
      },
      {
        "items": [
          {
            "response_type": "text",
            "text": "row 2 cell 2"
          }
        ]
      }
    ]
  }
 ]
}
```

iframe

Embeds content from an external website as an HTML `iframe` element.

Integration channel support

Web chat	Facebook
✔	✔

- Currently, the web chat integration ignores the `description` and `image_url` properties. Instead, the description and preview image are dynamically retrieved from the source at run time.

Fields

Name	Type	Description	Required?
response_type	string	`iframe`	Y
source	string	The URL of the external content. The URL must specify content that is embeddable in an HTML `iframe` element.	Y
title	string	The title to show before the embedded content.	N
description	string	The text of the description that accompanies the embedded content.	N
image_url	string	The URL of an image that shows a preview of the embedded content.	N
channel_options.chat.display	string	The way web chat renders the response type (`inline` or `panel`). The default value is `panel` for this response type.	N
channel_options.chat.dimensions.base_height	number	The base height (in pixels) to use to scale the content to a specific display size. This property works only when `display` is set to `inline`.	N

Different sites have varying restrictions for embedding content, and different processes for generating embeddable URLs. An embeddable URL is one that can be specified as the value of the `src` attribute of the `iframe` element.

For example, to embed an interactive map with Google Maps, you can use the Google Maps Embed API. (For more information, see The Maps Embed API overview.) Other sites have different processes for creating embeddable content.

For technical details about using `Content-Security-Policy: frame-src` to allow embedding of your website content, see CSP: frame-src.

Example

The following example embeds an iframe with a title and description.

```
{
  "generic":[
   {
     "response_type": "iframe",
```

```
    "source": "https://example.com/embeddable/example",
    "title": "Example iframe",
    "description": "An example of embeddable content returned as an iframe response.",
    "channel_options": {
      "chat": {
        "display": "inline",
        "base_height": 180
      }
    }
   }
  ]
}
```

image

Displays an image that is specified by a URL.

Integration channel support

Web chat	SMS	Slack	Facebook	WhatsApp	MS Teams
✔	✔	✔	✔	✔	✔

- Some channel integrations do not display image titles or descriptions.

Fields

Name	Type	Description	Required?
response_type	string	`image`	Y
source	string	The `https:` URL of the image. The specified image must be in `.jpg`, `.gif`, or `.png` format.	Y
title	string	The title to show before the image.	N
description	string	The text of the description that accompanies the image.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the image cannot be seen.	N

Example

This example displays an image with a title and descriptive text.

```
{
  "generic":[
   {
     "response_type": "image",
     "source": "https://example.com/image.jpg",
     "title": "Example image",
     "description": "An example image returned as part of a multimedia response."
   }
  ]
}
```

option

Use to show a set of options (such as buttons or a drop-down list) that users can choose from. The selected value is then sent to the assistant as user input. An `options` response is automatically defined when you choose the **Options** customer response type for a step. For more information, see Collecting information from your customers.

Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp	MS Teams

✔	✔	✔	✔	✔	✔	✔

- How options are presented varies by channel integration. The `preference` field is supported when possible, but not all channels support drop-down lists or buttons.

Fields

Name	Type	Description	Required?
response_type	string	`option`	Y
title	string	The title to show before the options.	Y
description	string	The text of the description that accompanies the options.	N
preference	string	The preferred type of control to display, if supported by the channel (`dropdown` or `button`).	N
options	list	A list of key-value pairs that specify options from which a user can choose.	Y
options[].label	string	The user-facing label for the option.	Y
options[].value	object	An object that defines the response that is sent to the watsonx Assistant service if the user selects the option.	Y
options[].value.input	object	An object that includes the message input corresponding to the option, including input text and any other field that is a valid part of a watsonx Assistant message. For more information, see the API Reference.	N

Example

This example presents two options (`Buy something` and `Exit`).

```
{
  "generic":[
   {
     "response_type": "option",
     "title": "Choose from the following options:",
     "preference": "button",
     "options": [
       {
         "label": "Buy something",
         "value": {
           "input": {
             "text": "Place order"
           }
         }
       },
       {
         "label": "Exit",
         "value": {
           "input": {
             "text": "Exit"
           }
         }
       }
     ]
   }
  ]
}
```

pause

Pauses before the next message to the channel, and optionally sends a "user is typing" event (for channels that support it).

Integration channel support

Web chat	Facebook	WhatsApp
✔	✔	✔

- With the phone integration, you can add a pause by including the SSML `break` element in the assistant output. For more information, see the Text to Speech documentation.

Fields

Name	Type	Description	Required?
response_type	string	`pause`	Y
time	int	How long to pause, in milliseconds.	Y
typing	Boolean	Whether to send the "user is typing" event during the pause. Ignored if the channel does not support this event.	N

Example

This example sends the "user is typing" event and pauses for 5 seconds.

```
{
   "generic":[
     {
       "response_type": "pause",
       "time": 5000,
       "typing": true
     }
   ]
}
```

speech_to_text

Sends a command to the Speech to Text service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

Integration channel support

Phone
✔

Fields

Name	Type	Description	Required?
response_type	string	`speech_to_text`	Y
command_info	object	Information specifying the command to send to the Speech to Text.	Y
command_info.type	string	The command to send (currently only the `configure` command is supported).	Y
command_info.parameters	object	See Applying advanced settings to the Speech to Text service	N

The `command_info.type` field can specify any of the following supported commands:

- `configure`: Dynamically updates the Speech to Text configuration. Configuration changes can be applied only to the next conversation turn, or for the rest of the session.

For information about how to use this command, see Applying advanced settings to the Speech to Text service.

Example

This example uses the `speech_to_text` response type with the `configure` command to change the language model from the Speech to Text service to Spanish, and to enable smart formatting.

```
{
  "generic": [
    {
      "response_type": "speech_to_text",
      "command_info": {
        "type": "configure",
        "parameters": {
          "narrowband_recognize": {
            "model": "es-ES_NarrowbandModel",
            "smart_formatting": true
          }
        }
      },
      "channels":[
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}
```

start_activities

Sends a command to a channel integration to start one or more activities that are specific to that channel. You can use this response type to restart any activity you previously stopped with the `stop_activities` response type.

Integration channel support

Phone
✔

Fields

Name	Type	Description	Required?
response_type	string	`start_activities`	Y
activities	list	A list of objects that identify the activities to start.	Y
activities[].type	string	The name of the activity to start.	Y

Currently, the following activities for the phone integration can be started:

- `speech_to_text_recognition`: Recognizes speech. Streaming audio to the Speech to Text service is resumed.
- `dtmf_collection`: Processes inbound DTMF signals.

Example

This example uses the `start_activities` response type to restart recognizing speech. Because this command is specific to the phone integration, the `channels` property specifies `voice_telephony` only.

```
{
  "generic": [
    {
      "response_type": "start_activities",
      "activities": [
        {
          "type": "speech_to_text_recognition"
```

```
      }
     ],
     "channels":[
      {
       "channel": "voice_telephony"
      }
     ]
    }
  ]
}
```

stop_activities

Sends a command to a channel integration to stop one or more activities that are specific to that channel. The activities remain stopped until they are restarted with the `start_activities` response type.

Integration channel support

Phone
✔

Fields

Name	Type	Description	Required?
response_type	string	`stop_activities`	Y
activities	list	A list of objects that identify the activities to stop.	Y
activities[].type	string	The name of the activity to stop.	Y

Currently, the following activities for the phone integration can be stopped:

- `speech_to_text_recognition`: Stops recognizing speech. All streaming audio to the Speech to Text service is stopped.
- `dtmf_collection`: Stops processing of inbound DTMF signals.

Example

This example uses the `stop_activities` response type to stop recognizing speech. Because this command is specific to the phone integration, the `channels` property specifies `voice_telephony` only.

```
{
  "generic": [
   {
    "response_type": "stop_activities",
    "activities": [
     {
      "type": "speech_to_text_recognition"
     }
    ],
    "channels":[
     {
      "channel":"voice_telephony"
     }
    ]
   }
  ]
}
```

table

Beta

The web chat support for new `table` response type represents the structured data in rows and columns, headers, and cells.

Integration channel support

Web chat
✔

Properties and Definitions

Property	Description	Type	Required
title	The title of the table.	String	No
description	A brief description of the table.	String	No
headers	Array of column headers.	Array<String, Number>	Yes
rows	Array of rows, each containing an array of cells.	Array<Object>	Yes
rows[].cells	Data for each cell in the row.	Array<String, Number>	Yes

Note: Each row must have the same number of cells as the headers. A mismatch between cells and headers will cause the web chat to throw an error when it attempts to render the table.

Example

This example displays structured data in a table.

```
{
   "generic": [
    {
      "response_type": "table",
      "title": "Data Table",
      "description": "A table with data",
      "headers": ["Column 1", "Column 2"],
      "rows": [
        {
          "cells": ["Row 1, Column 1", "Row 1, Column 2"]
        },
        {
          "cells": ["Row 2, Column 1", "22"]
        }
      ]
    }
  ]
}
```

text

Displays text (or reads it aloud, for the phone integration). To add variety, you can specify multiple alternative text responses. If you specify multiple responses, you can choose to rotate sequentially through the list, choose a response randomly, or output all specified responses.

Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✔	✔	✔	✔	✔	✔

Fields

Name	Type	Description	Required?
response_type	string	text	Y

values	list	A list of one or more objects that define a text response.	Y
values.[*n*].text_expression	object	An object that describes the text of the response.	N
values.[*n*].text_expression.concat	list	A list of objects that form components of the text response. These objects can include scalar text strings and references to variables.	N
selection_policy	string	How a response is selected from the list, if more than one response is specified. The possible values are `sequential`, `random`, and `multiline`.	N
delimiter	string	The delimiter to output as a separator between responses. Used only when `selection_policy=multiline`. The default delimiter is newline ( ).	N

Example

This example displays a greeting message to the user.

```
{
  "generic": [
    {
      "response_type": "text",
      "values": [
        {
          "text_expression": {
            "concat": [
              {
                "scalar": "Hi, "
              },
              {
                "variable": "step_472"
              },
              {
                "scalar": ". How can I help you?"
              }
            ]
          }
        }
      ],
      "selection_policy": "sequential"
    }
  ]
}
```

text_to_speech

Sends a command to the Text to Speech service instance used by the phone integration. These commands can dynamically change the configuration or behavior of the service during a conversation.

Integration channel support

Phone
✔

Fields

Name	Type	Description	Required?
response_type	string	`text_to_speech`	Y

command_info	object	Information specifying the command to send to the Text to Speech.	Y
command_info.type	string	The command to send (`configure`, `disable_barge_in`, or `enable_barge_in`).	Y
command_info.parameters	object	See Applying advanced settings to the Text to Speech service	N

The `command_info.type` field can specify any of the following supported commands:

- `configure`: Dynamically updates the Text to Speech configuration. Configuration changes can be applied only to the next conversation turn, or for the rest of the session.
- `disable_barge_in`: Disables speech barge-in so that playback from the phone integration is not interrupted when the customer speaks.
- `enable_barge_in`: Enables speech barge-in so that the customer can interrupt playback from the phone integration by speaking.

For detailed information about how to use each of these commands, see Applying advanced settings to the Text to Speech service.

Example

This example uses the `text_to_speech` response type with the `configure` command to change the voice used by the Text to Speech service.

```
{
  "generic": [
    {
      "response_type": "text_to_speech",
      "command_info": {
        "type": "configure",
        "parameters" : {
          "synthesize": {
            "voice": "en-US_LisaVoice"
          }
        }
      },
      "channels":[
        {
          "channel": "voice_telephony"
        }
      ]
    }
  ]
}
```

user_defined

A custom response type with any JSON data that the client or integration knows how to handle. For example, you might customize the web chat to display a special card, or build a custom application to format responses with a table or chart.

Note: The user-defined response type is not displayed unless the channel has code to handle it. For more information, see Customizing and developing.

Integration channel support

Web chat	Phone	SMS	Slack	Facebook	WhatsApp
✔	✔*	✔*	✔	✔	✔

- With the phone integration, the `user_defined` response type is used to send legacy commands (for example, `vgwActForceNoInputTurn` or `vgwActSendSMS`). For more information, see Handling phone interactions.
- With the SMS integration, the `user_defined` response type is used to send action commands (for example, `terminateSession` or `smsActSendMedia`).

Fields

Name	Type	Description	Required?

response_type	string	`user_defined`	Y
user_defined	object	An object that contains any data the client or integration knows how to handle. This object can contain any valid JSON data, but it cannot exceed a total size of 5000 bytes.	Y

Example

This example shows a generic example of a user-defined response. The `user_defined` object can contain any valid JSON data.

```
{
  "generic":[
   {
     "response_type": "user_defined",
     "user_defined": {
       "field_1": "String value",
       "array_1": [
         1,
         2
       ],
       "object_1": {
         "property_1": "Another string value"
       }
     }
   }
  ]
}
```

video

Displays a video that is specified by a URL.

Integration channel support

Web chat	SMS	Slack	Facebook	WhatsApp
✔	✔	✔	✔	✔

- Some channel integrations do not display video titles or descriptions.

Fields

Name	Type	Description	Required?
response_type	string	`video`	Y
source	string	The `https`: URL of the video. The URL can specify a video file or a streaming video on a supported hosting service.	Y
title	string	The title to show before the video.	N
description	string	The text of the description that accompanies the video.	N
alt_text	string	Descriptive text that can be used for screen readers or other situations where the video cannot be seen.	N
channel_options.chat.dimensions.base_height	number	The base height (in pixels) to use to scale the video to a specific display size.	N

The URL specified with the `source` property can be one of the following types:

- The URL of a video file in a standard format such as MPEG or AVI. In the web chat, the linked video renders as an embedded video player.

  HLS (`.m3u8`) and DASH (MPD) streaming videos are not supported.

- The URL of a video from a supported service. In the web chat, the linked video renders with the embeddable player for the hosting service.

  Specify the URL of the video that you want to view in your browser (for example, `https://www.youtube.com/watch?v=52bpMKVigGU`). The web chat automatically converts the URL to an embeddable form.

  You can embed videos that are hosted on the following services: - YouTube - Facebook - Vimeo - Twitch - Streamable - Wistia - Vidyard

Example

This example displays a video with a title and descriptive text.

```
{
  "generic":[
    {
      "response_type": "video",
      "source": "https://example.com/videos/example-video.mp4",
      "title": "Example video",
      "description": "An example video returned as part of a multimedia response."
    }
  ]
}
```

Supported foundation models

IBM® watsonx™ Assistant supports a diverse range of AI models for the capabilities within the product. For each model listed, you can learn more about their details so you can choose the model that fits your business needs.

Each model contains the reference for its **Model card** where you can get more information about technical specifications of the model.

Find information such as the supported languages, latest updates, and training data for the models available in their model card.

Foundation model lifecycle

As new versions of IBM foundation models are introduced, older versions are deprecated. Similarly, as newer and more effective models from other providers become available, less useful models are removed.

For more information on deprecated or withdrawn models, see Deprecated or withdrawn models.

granite-13b-chat-v2

Withdrawn

The Granite 13 Billion chat V2 (`granite-13b-chat-v2`) model is the chat-focused variant initialized from the pre-trained Granite Base 13 Billion Base V2 (`granite-13b-base-v2`) model. `granite-13b-base-v2` has been trained using over 2.5T tokens. IBM Generative AI Large Language Foundation Models are Enterprise-level English-language models trained with a large volume of data that has been subjected to intensive pre-processing and careful analysis.

Usage:

- Question answering
- Summarization
- Retrieval-Augmented Generation
- Classification
- Generation
- Extraction

Components that use this model:

- Base LLM.

Model card: ibm/granite-13b-chat-v2

granite-13b-instruct-v2

IBM Generative AI Large Language Foundation Models are Enterprise-level English-language models trained with a large volume of data that has been subjected to intensive pre-processing and careful analysis. The Granite 13 Billion Instruct V2.0 (granite.13b.instruct.v2) model is the instruction-tuned variant initialized from the pre-trained Granite Base 13 Billion Base V2.0 (granite.13b.base.v2) model. granite.13b.base.v2 is trained using over 2.5T tokens. The Granite family of models support all 5 language tasks (Q&A, Generate, Extract, Summarize, and Classify).

Usage:

- Question answering
- Summarization
- Retrieval-Augmented Generation
- Classification
- Generation
- Extraction

Components that use this model:

- Base LLM

Model card: ibm/granite-13b-instruct-v2

granite-3-8b-instruct

Granite-3.0-8B-Instruct is a 8B parameter model finetuned from Granite-3.0-8B-Base using a combination of open source instruction datasets with

permissive license and internally collected synthetic datasets. The model is designed to respond to general instructions and can be used to build assistants for multiple domains, including business applications.

Usage:

- Question answering
- Summarization
- Classification
- Generation
- Extraction
- Function calling

Components that use this model:

- Base LLM

Model card: ibm/granite-3-8b-instruct

granite-3-2b-instruct

Granite-3.0-2B-Instruct is a lightweight and open-source 2B parameter model fine tuned from Granite-3.0-2B-Base on a combination of open-source and proprietary instruction data with permissive license. The model is designed to respond to general instructions and can be used to build assistants for multiple domains, including business applications.

Usage:

- Question answering
- Summarization
- Classification
- Generation
- Extraction
- Function calling

Components that use this model:

- Base LLM

Model card: ibm/granite-3-2b-instruct

Preview runtime errors

You may see these system errors when testing your assistant in Preview.

Note: Some of these errors correspond to advanced development tasks.

Message	Description
No actions to interpret	Assistant has no content.
No action triggered	Error when an action is not found, for example, when one action specifies another action that isn't defined yet.
Conversation ended. Maximum of %d actions visited.	Limit is 20 per conversation.
Maximum number of step iterations for a single action	When a particular step is reached more than 50 times within an action.
Error when updating output in [%s]. The output is [%s]	When there was an error in updating the output, for example, when using a SpEL expression.
Error in context update	Error when setting a variable to an invalid ASEL expression.
Error when updating context with context of step %s. Step context is [%s].	Error when a runtime exception occurs while setting a context, for example, division by zero.
Multiple actions have the same title: %s	Actions need to have unique names.
Action %s is invalid because step order includes a cycle. Evaluation could result in an infinite loop. Check "next_step" property of each step in the action.	Step could get caught in an infinite loop and not end.
Expression evaluation error inside condition of %s. The syntax is valid, but cannot be evaluated. Condition defaulted to false. Check that objects in expression are not null or out of bounds. Error: %s	When an unexpected error occurs while checking a condition, for example, division by 0.
Provided expression [%s] cannot be evaluated as a number	ASEL error message
Provided expression [%s] cannot be evaluated as Boolean	ASEL error message
Provided expression [%s] cannot be evaluated as a string	ASEL error message
Provided expression [%s] cannot be evaluated	ASEL error message
Actions array contains an element that is not a JsonObject. Cannot build an internal representation.	Advanced programming message
Actions array contains elements with the same id (attribute 'action'). Cannot build an internal representation.	Advanced programming message
No target step %s found for jump-to resolver	Advanced programming message
For resolver of type jump_to the target step cannot be the same as the source step.	Advanced programming message

Built-in global variables

By writing expressions, you can access a set of global system variables that provide information about the conversation.

Each variable contains a JSON object that is taken from the `message` method input or output. For more information about these objects, see the API Reference information for the `message` method request and response.

> ⚠ **Important:** These variables are special system objects that require syntax different from the standard variable notation. To reference any of these values in an expression, use the variable name by itself (not including `?` or `{}` characters).

Variable	Description	Expression example
input	The `input` object from the most recent `message` request sent to the assistant.	input.text
output	The `output` object from the most recent `message` response. Currently, only `output.debug` is included.	output.debug.turn_events[0]

Built-in global system variables

Security certificates reference

You can use `Trust uploaded certificates and any certificates signed by a trusted authority` or `Trust uploaded certificates` to the following methods for connecting an assistant to an external service specified by a user:

- Custom extensions
- Conversational skills
- Premessage, Postmessage, and Log webhooks
- Search integrations (except Discovery search integration)

> **Note:** You don't need the security verification for Discovery search integration because the connections are managed internally and secured automatically.

Methods for choosing the Security certificates option

In the method examples, three possible Security certificate options are discussed with which you can achieve an optimal balance of security and convenience for an assistant with the following connection details.

Consider an assistant with the following connections:

- Three conversational skills each of which points to a different secure service. All three services are signed by the same private authority.
- Two custom extensions, each of which points to a secure service whose certificate is signed by the same publicly trusted authority.
- One search integration that connects to an Elasticsearch instance with a self-signed certificate (which is typical for Elasticsearch instances in IBM Cloud).
- One log webhook, which points to an insecure service (with a URL that starts with `http://` instead of `https://`).

> **Note:** Pointing to an insecure service is not recommended and your assistant can't verify the identity of such a service.

Method 1

1. Create a PEM file that contains:
   - The certificate for the private authority that signed the certificates for the three secure services that implement the conversational skills.
   - The self-signed certificate for the Elasticsearch instance.
2. Select **Trust uploaded certificates and any certificates signed by a trusted authority** .

Analysis

- More convenient option to use
- You don't need any additional effort to make the two custom extensions work because the services they point to have certificates that are signed by a publicly trusted authority
- You need only one certificate for the three conversational skills because they are all signed by the same authority and the certificate for that authority was included in the PEM file
- You can add more connections to services who certificates are also signed by the same authority without updating the PEM file

Method 2

1. Create a PEM file that contains:
   - The certificate for the private authority that signed the certificates for the three secure services that implement the conversational skills.
   - The certificate for the publicly trusted authority that signed the certificates for the two secure services that implement the custom extensions.
   - The self-signed certificate for the Elasticsearch instance.
2. Select **Trust uploaded certificates**.

Analysis

- Less convenient because the PEM file requires an extra entry for the authority certificate of custom extensions
- You must update the PEM file when other services that are signed by publicly trusted authorities are added in the future.
- Slightly more secure because it depends only on the one indicated publicly trusted authority being secure instead of depending on all publicly trusted authorities being secure
- The slight security advantage is not important to most users because all of the publicly trusted authorities are extremely secure

Method 3

1. Create a PEM file that contains:
    - The six certificates for the six secure services that the assistant connects to (three for conversational skills, two for custom extensions, and one for Elasticsearch).
    - The self-signed certificate for the Elasticsearch instance.
2. Select **Trust uploaded certificates**.

Analysis

- Less convenient because it requires more certificates in the PEM file
- Requires an update to the PEM file to add any additional service calls
- Slightly more secure because it does not assume that any public or private certifying authority is secure
- The additional security is not needed by most users because the public certifying authorities are extremely secure and the private certifying authority is controlled by you and hence can be secure.
- This option is available when you want the greatest security possible.

Your assistant does not verify the log webhook service in any of the methods because the log webhook points to an insecure server and the call to the log webhook continues to work. But the messages sent to and from the service lack encryption, making them vulnerable to imposter attacks.

The best way to address the vulnerabilities:

- Make the service more secure by implementing `https`.
- Update your assistant to call that `https` endpoint.
- Update the PEM file for your assistant, if needed.

Integration variables

By writing expressions, you can access integration variables, which are context variables that are specific to integrations (such as the web chat and phone integrations). All integration variables are contained in the `system_integrations` JSON object, which you can access using the following expression:

```
${system_integrations}
```

The `system_integrations` object has the following structure:

```
{
  "channel": {
    "name": "{channel_name}",
    "private": {
      "user": {
        "id": "{user_id}",
        "phone_number": "{phone_number}",
        "name": "{name}"
      }
    }
  },
  "chat": {...},
  "voice_telephony": {...},
  "text_messaging": {...},
  "whatsapp": {...},
  "slack": {...},
  "facebook": {...},
  "teams": {...}
}
```

channel

Always included. This object contains general information about the channel that is being used to communicate with the assistant.

Properties

Note: Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
`channel.name`	String	The name of the channel that is in use. One of the following values: • `Web chat` • `Phone` • `SMS` • `Whatsapp` • `Slack` • `Facebook Messenger` • `teams`
`channel.private.user.id`	String	The ID of the user who is interacting with the assistant through the channel. This ID is specific to the channel and might be different from the user ID watsonx Assistant uses for billing purposes. For more information, see Channel user IDs.
`channel.private.user.phone_number`	String	The phone number associated with the user. Set by the phone, SMS, and WhatsApp integrations.
`channel.private.user.name`	String	The name of the user who is interacting with the assistant through the channel. Set by the Microsoft Teams integration.
`channel.private.user.aadObjectId`	String	The Azure Active Directory (AAD) object ID of the user who is interacting with the assistant through the channel. Set by the Microsoft Teams integration.

Properties of the channel object

Example JSON

```
"channel": {
  {
    "name": "Web chat",
    "private": {
      "user": {
        "id": "anonymous_IBMuid-727c0302-6fd7-4abb-b7ee-f06b4bf30e99"
      }
    }
  }
}
```

Channel user ID

The `channel.private.user.id` property, which is set by the channel integration, specifies a user ID for the customer that is specific to the channel. The source of this user ID depends on the channel:

Channel	Channel user ID
Web chat	The user ID set by the web chat instance. For more information, see Managing user identity information in web chat.
Slack	The Slack member ID (for example, `U2147483697`).
Facebook	The Facebook sender ID (for example, `4310101122439797`).
Whatsapp	The customer's phone number.
Phone	The customer's phone number.
SMS with Twilio	The customer's phone number.
Teams	The Teams user ID (for example, `29:1CV2T75j3QUY-mQCQclk2...`) and provide the `aadObjectId` of the Teams user.

Sources of the channel user ID

chat

Included only if the web chat integration is in use.

Properties

Name	Type	Description
`browser_info.browser_name`	String	The browser name, such as `chrome`, `edge`, or `firefox`.
`browser_info.browser_version`	String	The browser version, such as `109.0.0`.
`browser_info.browser_OS`	String	The operating system of the customer's computer, such as `Mac OS`.
`browser_info.language`	String	The default locale code of the browser, such as `en-US`.
`browser_info.page_url`	String	The URL of the web page where the web chat is embedded, not including any query parameters or hashes.
`browser_info.screen_resolution`	String	The height and width of the browser window, such as `width: 1440, height: 900`.
`browser_info.user_agent`	String	The content of the HTTP `User-Agent` request header.

browser_info.client_ip_address	String	The IP address of the customer's computer.
browser_info.ip_address_list	Array	Ann array IP addresses specified by HTTP `X-Forwarded-For` request headers.

Properties of the chat object

Example JSON


```
"chat": {
  "browser_info": {
    "browser_name": "chrome",
    "browser_version": "109.0.0",
    "browser_OS": "Mac OS",
    "language": "en-US",
    "page_url": "https://us-south.assistant.watson.cloud.ibm.com/crn%3Av1%3Abluemix%3Apublic%3Aconversation%3Aus-south%3Aa%2Fc41400d63d91741a749091dc63574c2c%3Ab696c1e5-7316-4fb0-a61c-664438397e91%3A%3A/assistants/e344fcfe-506c-449f-a182-ebdefe4356ad/actions/actions/custom/edit/action_28584",
    "screen_resolution": "width: 1920, height: 1080",
    "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36",
    "client_ip_address": "65.191.135.254",
    "ip_address_list": [
      "65.191.135.254",
      " 104.99.56.143",
      "10.185.27.136",
      "172.30.226.64"
    ]
  }
},
"channel": {
  "name": "Web chat",
  "private": {
    "user": {
      "id": "anonymous_IBMuid-727c0302-6fd7-4abb-b7ee-f06b4bf30e99"
    }
  }
}
```


Example expressions

- This expression tests whether the customer is using the Chrome browser:

  ```
  $ ${system_integrations}.chat.browser_info.browser_name.equals("chrome")
  ```

  You might specify this expression as the value for a boolean session variable, which you could then use in a step condition. For example, if your website supports only Chrome, you could have a step conditioned on this variable being `false` and has the following output: `I see you aren't using the Chrome browser. Some features of our website work only on Chrome.`

- This expression tests whether the current page URL contains the string `payment.html`:

  ```
  $ ${system_integrations}.chat.browser_info.page_url.contains("payment.html")
  ```

  You might use this expression in a step condition in order to avoid telling customers to navigate to a page they are already viewing. For example, in an action for paying a bill, you could have a step conditioned on this variable being `false` and has the following output: `First, click **Pay bill** to navigate to the bill payment page.`

voice_telephony

Included only if the phone integration is in use.

The `voice_telephony` object contains both response and request properties. Request properties are values that are set by the phone integration and provide information about the call. Response properties are input values that you can modify in your actions to send logging data or to change configuration options for the call.

Request properties (set by the phone integration)

Note: Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
`sip_call_id`	String	The SIP call ID associated with the phone call.
`assistant_phone_number`	String	The phone number associated with with the watsonx Assistant end of the call.
`sip_custom_invite_headers`	Object	A user-defined array of key/value pairs containing SIP headers from the SIP `INVITE` request.
`private.user_phone_number`	String	The phone number from which the customer's call originated.
`private.sip_request_uri`	String	The inbound SIP request URI that initiated the phone call.
`private.sip_from_uri`	String	The URI from the `From` header of the SIP request.
`private.sip_to_uri`	String	The URI from the `To` header of the SIP request.
`final_utterance_timeout_occurred`	Boolean	Set to `true` when the final utterance timeout has been reached. This timeout can be configured by sending the `final_utterance_timeout_count` property.
`post_response_timeout_occurred`	Boolean	Set to `true` when the final utterance timeout has been reached. This timeout can be configured by sending the `post_response_timeout_count` property.

Request properties of the voice_telephony object

Example request JSON

```
$ "voice_telephony": {
  "private":{
    "user_phone_number":"+18595553456",
    "sip_request_uri":"sips:+18885557777@public.voip.us-east.assistant.watson.cloud.ibm.com",
    "sip_from_uri":"sips:+18565558576@twilio.com",
    "sip_to_uri":"sips:+18885557777@public.voip.us-east.assistant.watson.cloud.ibm.com"
  },
  "sip_call_id": "Aob2-2743-5678-1234",
  "assistant_phone_number":"+18885556789",
  "sip_custom_invite_headers": {
    "custom-header1": "123",
    "custom-header2": "456"
  }
}
```

Response properties (set by the assistant)

Name	Type	Description
`final_utterance_timeout_count`	Integer	The time (in milliseconds) to wait for a final utterance from the Speech to Text service. If no final utterance is received before the timeout occurs, the phone integration sends a message to the assistant with the `final_utterance_timeout_occurred` property set to `true`.
`post_response_timeout_count`	Integer	The time (in milliseconds) to wait for a new utterance after the last response is played. If no utterance is received before the timeout occurs, the phone integration sends a message to the assistant that includes the `post_response_timeout_occurred` property set to `true`.
`cdr_custom_data`	Object	A JSON object containing key/value pairs to be stored in the CDR record for the call. Each time this object is sent, its contents are merged with data sent previously during the call.
`turn_settings.timeout_count`	Integer	The time (in milliseconds) to wait for watsonx Assistant to finish processing each conversation turn.

Response properties of the voice_telephony object

Example response JSON

```
"voice_telephony" : {
  "post_response_timeout_count":10000,
  "final_utterance_timeout_count":30000,
  "turn_settings": {
    "timeout_count": 5000
  },
  "cdr_custom_data" : {
    "custom_data_1": "data 1",
    "custom_data_2": "data 2"
  }
}
```

text_messaging

Included only if the SMS with Twilio integration is in use.

Properties

Note: Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
assistant_phone_number	String	The phone number associated with with the watsonx Assistant end of the conversation.
private.user_phone_number	String	The phone number from which the customer's SMS message originated.

Properties of the text_messaging object

Example JSON

```
"text_messaging": {
  "private":{
    "user_phone_number":"+18595553456"
  },
  "assistant_phone_number":"+18885556789"
}
```

whatsapp

Included only if the WhatsApp integration is in use.

Properties

Note: Properties contained in the `private` object are treated as private variables, which are not included in logs.

Name	Type	Description
assistant_phone_number	String	The phone number associated with with the watsonx Assistant end of the conversation.
private.user_phone_number	String	The phone number from which the customer's WhatsApp message originated.

Properties of the whatsapp object

Example JSON

```
"whatsapp": {
  "private":{
    "user_phone_number":"+18595553456"
  },
```

```
  "assistant_phone_number":"+18885556789"
}
```

slack

Included only if the Slack integration is in use.

Properties

Name	Type	Description
team_id	String	The unique identifier of the Slack team.
channel_id	String	The unique identifier of the Slack channel.

Properties of the slack object

Example JSON

```
"slack": {
  "team_id":"T02F3KE542J",
  "channel_id":"C4K3KTTRD"
}
```

facebook

Included only if the Facebook integration is in use.

Properties

No additional properties.

Example JSON

```
"facebook": {}
```

teams

Included only if the Microsoft Teams integration is in use.

Properties

Name	Type	Description
conversation_id	String	The unique identifier of the Microsoft Teams conversation.

Properties of the teams object

Example JSON

```
"teams":{
  "conversation_id":"a:1ATy08jyGkPGy2QdKIrGZL5u_o6fIUVDRKeIZtkIUAkQDC23FC9S97f18i-UNl-
eISAfDWqoQeTbregvSE8jK0LNy6h9VssNcN3CsGG9guMiUB0EeSqxnnEFpAVzbkayR"
}
```

Expression language methods for actions

The watsonx Assistant expression language can be used to specify values that are independent of, or derived from, values that are collected in steps or stored in session variables. You can use an expression to define a step condition or to define the value of a session variable.

> **Note:** The watsonx Assistant expression language is based on the Spring Expression Language (SpEL), but with some important differences in syntax. For detailed background information about SpEL, see Spring Expression Language (SpEL).

You can use SpEL expressions in two ways:

- Defining a step condition. For more information, see Writing expressions.
- Assigning a value to a session variable. For more information, see Using variables to manage conversation information.

Referencing action variables

An action variable is created implicitly for any step that expects customer input, and the variable is bound to the step. To reference an action variable inside an expression, you must specify the step ID by using the format `${step_id}` (for example, `${step_771}`. To find the step ID for a step, select the step and then look at the end of the URL in your browser.

Referencing session variables

Session variables are created explicitly in the **Variables** section of the step. To reference a session variable inside an expression, you must specify the variable ID by using the format `${variable_id}` (for example, `${current_time}`. You can find the variable ID in the list of variables. (For more information, see Using variables to manage conversation information.)

When you are editing an expression, you can type `$` to see a list of variables you can reference. Select a variable from the list to automatically insert the step ID or variable ID.

Supported data types

Expressions can use atomic JSON types (such as `integer`, `string`, `number`, and `boolean`), and compound data types (such as JSON arrays (`[]`) and objects (`{}`). When you specify literal string values, you can use either single (`'`) or double (`"`) quotation marks.

Values that action steps collect from customers use customer response types such as date, time, currency, or percent. These values are stored as JSON objects in the following format:

```
{
  "system_type": "{system_type}",
  "value": "{value}"
}
```

where `{system_type}` is one of the following types:

- `time`
- `percentage`
- `currency`

Date and Time methods

Several methods are available to work with date and time values.

now(String timezone)

The `now()` method returns the current date and time for a specified time zone, in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`:

```
$ now('Australia/Sydney').
```

In this example, if the current date and time is `2021-11-26 11:41:00`, the returned string is `2021-11-26 21:41:00 GMT+10.00` or `2021-11-26 21:41:00 GMT+11.00` depending on Daylight Saving Time.

The output string format change is applicable to date and time calculation methods as well. For example, if the `<date>` string used is in the format `yyyy-MM-dd HH:mm:ss`, such as when you use the method `today()`, then the output is in the same format (`yyyy-MM-dd HH:mm:ss`). However, if the `<date>` string is in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`, such as when you use the method `now()`, then the output is in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`.

.reformatDateTime(String format)

Formats date and time strings for output. The parameter is a format string that specifies how the date or time value is formatted. The format string must be specified by using the Java SimpleDateFormat syntax.

This method returns a string that is formatted according to the specified format:

- `MM/dd/yyyy` for 12/31/2016
- `h a` for 10pm

To return the day of the week:

- `EEEE` for Tuesday
- `E` for Tue
- `u` for day index (1 = Monday, ..., 7 = Sunday)

For example, this expression returns the value 17:30:00 as `5:30 PM`:

```
${system_current_date}.reformatDateTime('h:mm a')
```

Note: If the input string includes only a time, the default date `1970-01-01` is used in the output. If the input string includes only a date, the default time `12 AM (00:00)` is used.

.before(String date/time)

- Determines whether a date and time value is before the specified date and time argument, as in this example:

```
$ ${system_current_date}.before('2021-11-19')
```

You can compare a date with another date, or a time with another time. You can also compare a time with a date and time value, in which case the date is ignored and only the times are compared. Any other comparisons of mismatched values (for example, comparing a date with a time) return `false`, an exception is logged in `output.debug.log_messages` (which you can see in the assistant preview or in API responses).

.after(String date/time)

- Determines whether the date and time value is after the date and time argument.

.sameMoment(String date/time)

- Determines whether the date and time value is the same as the date and time argument.

.sameOrAfter(String date/time)

- Determines whether the date and time value is after or the same as the date and time argument.
- Analogous to `.after()`.

.sameOrBefore(String date/time)

- Determines whether the date and time value is before or the same as the date and time argument.

Date and time calculations

Use the following methods to calculate a date.

Method	Description
`<date>.minusDays(_n_)`	Returns the date of the day *n* days before the specified date.
`<date>.minusMonths(_n_)`	Returns the date of the day *n* months before the specified date.
`<date>.minusYears(_n_)`	Returns the date of the day *n* years before the specified date.

<date>.plusDays(_n_)	Returns the date of the day *n* days after the specified date.
<date>.plusMonths(_n_)	Returns the date of the day *n* months after the specified date.
<date>.plusYears(n)	Returns the date of the day *n* years after the specified date.

Date calculations

where `<date>` is specified in the format `yyyy-MM-dd` or `yyyy-MM-dd HH:mm:ss`.

For example, to get tomorrow's date, specify the following expression:

```
${system_current_date}.plusDays(1)
```

Use the following methods to calculate a time.

Method	Description
<time>.minusHours(_n_)	Returns the time *n* hours before the specified time.
<time>.minusMinutes(_n_)	Returns the time *n* minutes before the specified time.
<time>.minusSeconds(_n_)	Returns the time *n* seconds before the specified time.
<time>.plusHours(_n_)	Returns the time *n* hours after the specified time.
<time>.plusMinutes(_n_)	Returns the time *n* minutes after the specified time.
<time>.plusSeconds(_n_)	Returns the time *n* seconds after the specified time.

Time calculations

where `<time>` is specified in the format `HH:mm:ss`.

For example, to get the time one hour from now, specify the following expression:

```
now().plusHours(1)
```

Working with time spans

To show a response based on whether today's date falls within a certain time span, you can use a combination of time-related methods. For example, if you run a special offer during the holiday season every year, you might want to check whether today's date falls between November 25 and December 24 of this year.

First, define the dates of interest as session variables. In the following start and end date session variable expressions, the date is being constructed by concatenating the dynamic current year value with hardcoded month and day values:

```
start_date = now().reformatDateTime('Y') + '-12-24'
end_date = now().reformatDateTime('Y') + '-11-25'
```

Then, in a step condition, you can indicate that you want to show the response only if the current date falls between the start and end dates that you defined as session variables:

```
now().after(${start_date}) && now().before(${end_date})
```

java.util.Date support

In addition to the built-in methods, you can use standard methods of the `java.util.Date` class.

For example, to get the date of the day that falls a week from today, you can use the following syntax:

```
new Date(new Date().getTime() + (7 * (24*60*60*1000L)))
```

This expression first gets the current date in milliseconds (since January 1, 1970, 00:00:00 GMT). It also calculates the number of milliseconds in 7 days (`(24*60*60*1000L)` represents one day in milliseconds). It then adds 7 days to the current date. The result is the full date of the day that falls a week from today (for example, `Fri Jan 26 16:30:37 UTC 2018`).

Number methods

These methods help you get and reformat number values.

For information about recognizing numbers in customer responses, see Choosing a response type.

If you want to change the decimal placement for a number (for example, to reformat a number as a currency value), see the String format() method.

toDouble()

Converts the object or field to the Double number type. You can call this method on any object or field. If the conversion fails, `null` is returned.

toInt()

Converts the object or field to the Integer number type. You can call this method on any object or field. If the conversion fails, `null` is returned.

toLong()

Converts the object or field to the Long number type. You can call this method on any object or field. If the conversion fails, `null` is returned.

To specify a Long number type in a SpEL expression, you must append an `L` to the number to identify it as such (for example, `5000000000L`). This syntax is required for any numbers that do not fit into the 32-bit Integer type. Numbers that are greater than 2^31 (2,147,483,648) or less than -2^31 (-2,147,483,648) are considered Long number types. Long number types have a minimum value of -2^63 and a maximum value of 2^63-1 (or 9,223,372,036,854,775,807).

Standard math

Use SpEL expressions to define standard math equations, where the operators are represented by using these symbols:

Arithmetic operation	Symbol
addition	+
division	/
multiplication	*
subtraction	-

Standard math

java.lang.Math()

You can use the functions of the java.lang.Math class to perform basic numeric operations.

You can use the Class methods, including:

- `max()`:

```
T(Math).max(${step_297},${step_569})
```

- `min()`:

```
T(Math).min(${step_297},${step_569})
```

- `pow()`:

```
T(Math).pow(${step_297}.toDouble(),2.toDouble())
```

See the java.lang.Math reference documentation for information about other methods.

java.util.Random()

Returns a random number. You can use any of the following syntax options:

- To return a random Boolean value (`true` or `false`), use `new Random().nextBoolean()` .
- To return a random double number between 0 (included) and 1 (excluded), use `new Random().nextDouble()`
- To return a random integer between 0 (included) and a number you specify, use `new Random().nextInt(_n_)` , where *n* is 1 greater than the top of the number range you want. For example, if you want to return a random number between 0 and 10, specify `new Random().nextInt(11)` .
- To return a random integer from the full integer value range (-2147483648 to 2147483648), use `new Random().nextInt()` .

For example, you might create step that is executed only for a randomly selected subset of customers. The following step condition would mean that the step has a 50% chance of executing:

```
new Random().nextInt(2) == 1
```

See the java.util.Random reference documentation for information about other methods.

You can also use standard methods of the following classes:

- `java.lang.Byte`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Double`
- `java.lang.Short`
- `java.lang.Float`

String methods

These methods help you work with text.

> **Note:** For details about the syntax to use in methods that involve regular expressions, see RE2 Syntax reference.

String.append(Object)

This method appends an input object (as a string) to a string, and returns a modified string.

```
${step_297}.append('next text')
```

String.contains(String)

This method returns `true` if the action variable or session variable contains a substring, which is useful in conditions.

```
${step_297}.contains('Yes')
```

String.endsWith(String)

This method returns `true` if the string ends with the input substring.

```
${step_297}.endsWith('?')
```

String.equals(String)

This method returns `true` if the specified string equals the action variable or session variable.

```
${step_297}.equals('Yes')
```

String.equalsIgnoreCase(String)

This method returns `true` if the specified string equals the action variable or session variable irrespective of case.

```
${step_297}.equalsIgnoreCase('Yes')
```

String.extract(String regexp, Integer groupIndex)

This method returns a string from the input that matches the specified regular expression group pattern. It returns an empty string if no match is found.

> **Note:** This method is designed to extract matches for different regex pattern groups, not different matches for a single regex pattern. To find different matches, see the getMatch() method.

In this example, the action variable is saving a string that matches the regex pattern group that you specify. In the expression, two regex patterns groups are defined, each one enclosed in parentheses. Inherent is a third group that is composed of the two groups. This is the first (groupIndex 0) regex group; it matches with a string that contains the full number group and text group. The second regex group (groupIndex 1) matches with the first occurrence of a number group. The third group (groupIndex 2) matches with the first occurrence of a text group after a number group.

```
${step_297}.extract('([\d]+)(\b [A-Za-z]+)', <n>)
```

If action variable contains:

```
Hello 123 this is 456.
```

the results are as follows:

- When `<n>` = `0`, the value is `123 this`.
- When `<n>` = `1`, the value is `123`.
- When `<n>` = `2`, the value is `this`.

String.find(String regexp)

This method returns `true` if any segment of the string matches the input regular expression. You can call this method against a JSONArray or JSONObject element, and it converts the array or object to a string before it makes the comparison.

For example, if the action variable `${step_297}` collects the string `Hello 123456`, then the following expression returns `true`:

```
${step_297}.find('^[^\d]*[\d]{6}[^\d]*$')
```

The condition is `true` because the numeric portion of the input text matches the regular expression `^[^\d]*[\d]{6}[^\d]*$`.

String.getMatch(String regexp, Integer matchIndex)

This method returns a string that matches the occurrence of the specified regular expression pattern. This method returns an empty string if no match is found.

As matches are found, they are added to what you can think of as an array of matches. Because the array element count starts at 0, if you want to return the third match, you would specify 2 as the `matchIndex` value. For example, if you enter a text string with three words that match the specified pattern, you can return the first, second, or third match only by specifying its index value.

For example, the following example is looking for a group of numbers in an action variable.

```
${step_297}.getMatch('([\d]+)', 1)
```

If the action variable `${step_297}` contains the string `hello 123 i said 456 and 8910`, this expression returns `456`.

String.isEmpty()

This method returns `true` if the string is an empty string, but not `null`, as in this example:

```
${step_297}.isEmpty()
```

String.length()

This method returns the character length of the string, as in this example:

```
${step_297}.length()
```

If the action variable `${step_297}` contains the string `Hello`, this expression returns 5.

String.matches(String regexp)

This method returns `true` if the string matches the input regular expression, as in the example:

```
${step_297}.matches('^Hello$')
```

If the action variable `${step_297}` contains the string `Hello`, this expression evaluates to `true`.

String.startsWith(String)

This method returns `true` if the string starts with the specified substring, as in this example:

```
${step_297}.startsWith('What')
```

If the action variable `${step_297}` contains the string `What is your name?`, this expression returns `true`.

String.substring(Integer beginIndex, Integer endIndex)

This method returns a substring beginning with the character at `beginIndex` and ending with the character before `endIndex`. (The `endIndex` character itself is not included in the substring.) The index values are zero-based, so the first character in the string is at index 0.

This example returns a substring that begins at index 5 (which is the sixth character), and continuing to the end of the string:

```
${step_297}.substring(5, ${step_297}.length())
```

If the action variable `${step_297}` contains the string `This is a string.`, this expression returns `is a string.`

String.toJson()

This method parses a string that contains JSON data and returns a JSON object or array, as in this example:

```
${json_var}.toJson()
```

If the session variable `${json_var}` contains the following string:

```
"{ \"firstname\": \"John\", \"lastname\": \"Doe\" }"
```

the `toJson()` method returns the following object:

```
{
  "firstname": "John",
  "lastname": "Doe"
}
```

String.toLowerCase()

This method returns the specified string that is converted to lowercase letters, as in this example:

```
${step_297}.toLowerCase()
```

If the action variable `${step_297}` contains the string `This is A DOG!`, this expression returns the string `this is a dog!`.

String.toUpperCase()

This method returns the original string that is converted to uppercase letters, as in this example:

```
${step_297}.toUpperCase()
```

If the action variable `${step_297}` contains the string `hi there`, this method returns the string `HI THERE`.

String.trim()

This method trims any spaces at the beginning and end of a string and returns the modified string, as in this example:

```
${step_297}.trim()
```

If the action variable `${step_297}` contains the string `something is here`, this method returns the string `something is here`.

java.lang.String support

In addition to the built-in methods, you can use standard methods of the `java.lang.String` class.

java.lang.String.format()

You can apply the standard Java String `format()` method to text. For information about the syntax to use, see Format String Syntax.

This example takes three decimal integers (1, 1, and 2) and adds them to a sentence:

```
T(java.lang.String).format('%d + %d equals %d', 1, 1, 2)
```

The resulting string is `1 + 1 equals 2`.

This example changes the decimal placement for a number that is collected by a step:

```
T(String).format('%.2f',${step_297})
```

If the `${step_297}` variable that needs to be formatted in US dollars is 4.5, the resulting string is `4.50`.

Array methods

These methods help you work with arrays.

Array.add(value...)

This method adds one or more new values to the array and returns `true` if the operation is successful.

```
${Items}.add('four', 'five')
```

If `Items` is `['one', 'two', 'three']`, this example updates it in place to become `['one', 'two', 'three', 'four', 'five']`.

Array.addAll(Array array)

This method adds one array to another and returns `null`.

```
${Items}.addAll(${New_items})
```

If `Items` is `['one', 'two', 'three']` and `New_items` is `['four', 'five', 'six']`, this example updates `Items` in place to become `['one', 'two', 'three', 'four', 'five', 'six']`.

Array.append(value...)

This method appends one or more new values to the array and returns the result as a new array. The original array is not modified.

```
${Items}.append('four', 'five')
```

If `Items` is `['one', 'two', 'three']`, this example returns the new array `['one', 'two', 'three', 'four', 'five']`.

Array.clear()

This method removes all values from the array and returns null.

```
$ ${Items}.clear()
```

After this expression is evaluated, `Items` is an empty array (`[]`).

Array.contains(value)

This method returns `true` if the array contains an item that is exactly equal to the input value. The specified value can be a string or a number.

```
$ ${Items}.contains(123)
```

If `Items` is `[123, 456, 789]`, this example returns `true`.

Array.containsIgnoreCase(value)

This method returns `true` if the array contains an item that is equal to the input value. Strings are matched regardless of whether the value is specified in uppercase or lowercase letters. The specified value can be a string or a number.

```
$ ${Items}.contains('two')
```

This example returns `true` if the `Items` array contains any capitalization of the string `two` (for example, `TWO` or `Two` would also match).

Array.filter(temp_var, "temp_var.property operator comparison_value")

Filters an array by comparing each array element to a value you specify, returning a new array that contains only the matching elements.

The filter expression consists of the following values:

- `temp_var`: An arbitrary name for a temporary variable that is used to hold each array element as it is evaluated. For example, if the original array contains objects that describe cities, you might use `city` as the temporary variable name.
- `property`: The element property that you want to filter on. This must be a property of the elements in the source array. Specify the property as a property of `temp_var`, by using the syntax `temp_var.property`. For example, if `latitude` is a valid property name for the source elements, you might specify the property as `city.latitude`.
- `operator`: The operator to use to compare the property value to the comparison value. You can use any of the following operators:

Operator	Description
==	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
!=	Is not equal to

Supported filter operators

- `comparison_value`: The value that you want to compare each array element property value to. You can specify a literal value or reference a variable.

Filter examples

For example, you might have an array of objects that contain city names and their population numbers:

```
[
  {
    "name":"Tokyo",
    "population":13988129

  },
  {
    "name":"Rome",
    "population":2860009

  },
  {
    "name":"Beijing",
    "population":21893095

  },
  {
    "name":"Paris",
```

```
    "population":2165423

  }
]
```

If the source array is stored in a variable that is called `${cities}`, the following expression returns a smaller array that contains only cities with populations greater than 5 million:

```
${cities}.filter("city", "city.population > 5000000")
```

The expression returns the following filtered array:

```
[
  {
    "name":"Tokyo",
    "population":13988129

  },
  {
    "name":"Beijing",
    "population":21893095
  }
]
```

Instead of a hardcoded comparison value, you might also filter based on a dynamic value that is stored in a variable. This example filters by using a population value that is specified by a customer response in a previous step:

```
${cities}.filter("city", "city.population > ${step_123}")
```

Tip: When you compare number values, be sure to set the context variable that is involved in the comparison to a valid value before the filter method is triggered. Note that `null` can be a valid value if the array element you are comparing it against might contain it.

Array.get(Integer index)

This method returns the item from the array that is at the specified index position. Arrays are zero-indexed, meaning that the first item in the array is at index position `0`.

```
$ ${Items}.get(1)
```

If `Items` is `['one', 'two', 'three']`, this example returns `two`.

The `get()` method is an alternative to using brackets (`[]`) to retrieve an item from an array. The following example is also valid and returns the same result:

```
$ ${Items}[1]
```

Note: If you are using a value that is specified by a customer to choose an item from an array, you might need to subtract 1 to convert to a zero-indexed value. For example, you might use an expression like `${Items}.get(${step_123} - 1)` to retrieve the intended value.

Array.getRandomItem()

This method returns a randomly chosen item from the array.

```
$ ${Items}.getRandomItem()
```

If `Items` is `['one', 'two', 'three']`, this example returns `one`, `two`, or `three`, on a random basis.

Array.indexOf(value)

This method returns the index position of the first occurrence of the input value in the array, or `-1` if the array does not contain the input value. The specified value can be a string or a number.

```
$ ${Items}.indexOf(`two`)
```

If `Items` is `['one', 'two', 'three']`, this example returns the integer `1` (indicating the second position in the zero-indexed array).

Array.join(String delimiter)

This method joins all values in this array to a string. Values are converted to string and delimited by the input delimiter.

For example, you might use a variable that is named `pizza_toppings` that contains the array `["pepperoni", "ham", "mushrooms"]`. The following expression converts this array into the string `pepperoni, ham, mushrooms`:

```
$ ${toppings_array}.join(', ')
```

If you use that expression to define the value of a variable, you can then reference that variable in your assistant output to create a human-readable message (for example, `You have selected the following toppings: pepperoni, ham, mushrooms`).

JSONArray.joinToArray(template, retainDataType)

This method extracts information from each item in the array and builds a new array that is formatted according to the template you specify. The template can be a string, a JSON object, or an array. The method returns an array of strings, an array of objects, or an array of arrays, depending on the type of template.

This method is useful for formatting information as a string that you can return as part of the output of a step, or for transforming data into a different structure so you can use it with an external API.

In the template, you can reference values from the source array by using the following syntax:

```
$ %e.{property}%
```

where `{property}` represents the name of the property in the source array.

For example, suppose that your assistant stores an array that contains flight details in a session variable. The stored data might look like this:

```
"flights": [
    {
      "flight": "AZ1040",
      "origin": "JFK",
      "carrier": "Alitalia",
      "duration": 485,
      "destination": "FCO",
      "arrival_date": "2019-02-03",
      "arrival_time": "07:00",
      "departure_date": "2019-02-02",
      "departure_time": "16:45"
    },
    {
      "flight": "DL1710",
      "origin": "JFK",
      "carrier": "Delta",
      "duration": 379,
      "destination": "LAX",
      "arrival_date": "2019-02-02",
      "arrival_time": "10:19",
      "departure_date": "2019-02-02",
      "departure_time": "07:00"
    },
    {
      "flight": "VS4379",
      "origin": "BOS",
      "carrier": "Virgin Atlantic",
      "duration": 385,
      "destination": "LHR",
      "arrival_date": "2019-02-03",
      "arrival_time": "09:05",
      "departure_date": "2019-02-02",
      "departure_time": "21:40"
    }
]
```

To build an array of strings that describe these flights in a user-readable form, you might use the following expression:

```
$ ${Flight_data}.joinToArray("Flight %e.flight% to %e.destination%", true)
```

This expression would return the following array of strings: `["Flight AZ1040 to FCO","Flight DL1710 to LAX","Flight VS4379 to LHR"]`.

The optional `retainDataType` parameter specifies whether the method preserves the data type of all input values in the returned array. If `retainDataType` is set to `false` or omitted, in some situations, strings in the input array might be converted to numbers in the returned array. For example, if the selected values from the input array are `"1"`, `"2"`, and `"3"`, the returned array might be `[ 1, 2, 3 ]`. To avoid unexpected type conversions, specify `true` for this parameter.

Complex templates

A more complex template might contain formatting that displays the information in a legible layout. For a complex template, you might want to store the template in a session variable, which you can then pass to the `joinToArray` method instead of a string.

For example, this complex template contains a subset of the array elements, adding labels and formatting:

```
<br/>Flight number: %e.flight% <br/> Airline: %e.carrier% <br/> Departure date: %e.departure_date% <br/> Departure time: %e.departure_time% <br/> Arrival time: %e.arrival_time% <br/>
```

Note: Make sure any formatting that you use in your template is supported by the channel integration that displays the assistant output.

If you create a session variable that is named `Template`, and assign this template as its value, you can then use that variable in your expressions:

```
$ ${Flight_data}.joinToArray(${Template})
```

At run time, the response would look like this:

```
Flight number: AZ1040
Airline: Alitalia
Departure date: 2019-02-02
Departure time: 16:45
Arrival time: 07:00

Flight number: DL1710
Airline: Delta
Departure date: 2019-02-02
Departure time: 07:00
Arrival time: 10:19

Flight number: VS4379
Airline: Virgin Atlantic
Departure date: 2019-02-02
Departure time: 21:40
Arrival time: 09:05
```

JSON templates

Instead of a string, you can define a template as a JSON object, which provides a way to standardize the formatting of information from different systems, or to transform data into the format required for an external service.

In this example, a template is defined as a JSON object that extracts flight details from the elements that are specified in the array that is stored in the `Flight data` session variable:

```
{
  "departure": "Flight %e.flight% departs on %e.departure_date% at %e.departure_time%.",
  "arrival": "Flight %e.flight% arrives on %e.arrival_date% at %e.arrival_time%."
}
```

Using this template, the `joinToArray()` method returns a new array of objects with the specified structure.

Array.remove(Integer index)

This method removes the item in the specified index position from the array, and returns the updated array.

```
$ ${Items}.remove(1)
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'three']`. The original `Items` array is also modified in place.

Array.removeValue(value)

This method removes the first occurrence of the specified value from the array, and returns the updated array. The specified value can be a string or a number.

```
$ ${Items}.removeValue('two')
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'three']`. The original `Items` array is also modified in place.

Array.set(Integer index, value)

This method replaces the item in the specified index position with the specified value, and returns the updated array.

```
$ ${Items}.set(2,'five')
```

If `Items` is `['one', 'two', 'three']`, this example returns `['one', 'two', 'five']`. The original `Items` array is also modified in place.

Array.size()

This method returns the number of items in the array as an integer.

```
$ ${Items}.size()
```

If `Items` is `['one', 'two', 'three']`, this example returns `3`.

Array.sort()

This method does an in-place sorting and returns the sorted array. The default parameter is `ascending`. You can specify `descending` to change the sort order. Any other parameter entry is ignored and a log error appears.

```
$ ${Items}.sort("ascending")
```

The method compares numbers and strings. A number is always smaller than a string. Any other type is converted to string by default to compare.

For example:

Original array	Sorted array
[2,1,3,5,4,3,2]	[1,2,2,3,3,4,5]
["Banana", "Orange", "Apple", "Mango"]	["Apple", "Banana", "Mango", "Orange"]
[3, 2, 4, "1", "10", "12", "Banana", "Orange", 0, "Apple", "Mango"]	[0, 2, 3, 4, "1", "10", "12", "Apple", "Banana", "Mango", "Orange"]

Sorting examples

Array.transform()

The `Array.transform()` method is used with the session_history variable only. You can transform the output of the variable to match a specific generative AI system.

Table: Signatures for chat formats shows signatures that you can use for the different chat formats:

Particulars	OpenAI	Google PaLM2	Llama2
Signature	`transform(String rolePrefix, String userPrefix, String assistantPrefix, optional Boolean currentAction=false)`	`transform(String rolePrefix, String userPrefix, String assistantPrefix, optional Boolean currentAction=false)`	`transform(optional String systemPrompt, optional Boolean currentAction=false)`

Customer message format	{$rolePrefix: $userPrefix, "content": $content}	{$rolePrefix: $userPrefix, "content": $content}	<s>[INST] <<SYS>>{{ $systemPrompt }} <</SYS>>{{ $user_content }} [/INST] {{ $assistant_content }} </s> <s>[INST] {{ $user_content }} [/INST]
Assistant message format	{$rolePrefix: $assistantPrefix, "content": $content}	{$rolePrefix: $assistantPrefix, "content": $content}	NA
Example	${system_session_history}.transform("role", "user", "assistant")	${system_session_history}.transform("author", "USER", "AI")	${system_session_history}.transform("<your system prompt>")

Table: Signatures for chat formats

If `currentAction` is true:

Assistant uses	Description
Actions only	The transform excludes all messages where `n`is true, which indicates that a customer question triggered a new base action.
Dialog only	`currentAction` is ignored and the transform includes the entire contents of the `session history` variable.
Dialog and actions	The transform includes everything in the `session_history` variable since the most recent start of an action, regardless of whether any dialog nodes are triggered.

If currentAction is true

The `n : true` flags are not included in the output of transform.

Expression language methods for dialog

You can process values that are extracted from user utterances that you want to reference in a context variable, condition, or elsewhere in the response.

Where to use the expression syntax

To expand variable values inside other variables, or apply methods to output text or context variables, use the `<? expression ?>` expression syntax. For example:

- Referencing a user's input from a dialog node text response

  ```
  You said <? input.text ?>.
  ```

- Incrementing a numeric property from the JSON editor

  ```
  "output":{"number":"<? output.number + 1 ?>"}
  ```

- Checking for a specific entity value from a dialog node condition

  ```
  @city.toLowerCase() == 'paris'
  ```

- Checking for a specific date range from a dialog node response condition

  ```
  @sys-date.after(today())
  ```

- Adding an element to a context variable array from the context editor

Context variable name	Context variable value
toppings	<? context.toppings.append('onions') ?>

Context variable array

You can use SpEL expressions in dialog node conditions and dialog node response conditions also.

⚠ **Important:** When a SpEL expression is used in a node condition, the surrounding `<? ?>` syntax is not required.

The following sections describe methods that you can use to process values. They are organized by data type:

- Arrays
- Date and Time
- Numbers
- Objects
- Strings

Arrays

You cannot use these methods to check for a value in an array in a node condition or response condition within the same node in which you set the array values.

JSONArray.addAll(JSONArray)

This method appends one array to another.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives"],
    "more_toppings": ["mushroom","pepperoni"]
  }
}
```

Make this update:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.addAll($more_toppings) ?>"
  }
}
```

Result: The method itself returns `null`. However, the first array is updated to include the values from the second array.

```
{
  "context": {
    "toppings_array": ["onion", "olives", "mushroom", "pepperoni"]
  }
}
```

JSONArray.append(object)

This method appends a new value to the JSONArray and returns the modified JSONArray.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives"]
  }
}
```

Make this update:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.append('ketchup', 'tomatoes') ?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ketchup", "tomatoes"]
  }
}
```

JSONArray.clear()

This method clears all values from the array and returns null.

Use the following expression in the output to define a field that clears an array that you saved to a context variable ($toppings_array) of its values.

```
{
  "output": {
    "array_eraser": "<? $toppings_array.clear() ?>"
  }
}
```

Then, if you reference the $toppings_array context variable, it returns '[]' only.

JSONArray.contains(Object value)

This method returns true if the input JSONArray contains the input value.

For this dialog runtime context that is set by a previous node or external application:

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

```
    "toppings_array": ["onion", "olives", "ham"]
  }
}
```

Dialog node or response condition:

```
$toppings_array.contains('ham')
```

Result: `true` because the array contains the element `ham`.

JSONArray.containsIgnoreCase(Object value)

This method returns `true` if the input JSONArray contains the input value, regardless of whether the value is specified in uppercase or lowercase letters.

For this dialog runtime context that is set by a previous node or external application:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ham"]
  }
}
```

Dialog node or response condition:

```
$toppings_array.containsIgnoreCase('HAM')
```

Result: `true` because the array contains the element `ham` and the case is ignored.

JSONArray.containsIntent(String intent_name, Double min_score, [Integer top_n])

This method returns `true` if the `intents` JSONArray specifically contains the specified intent, and that intent has a confidence score that is equal to or higher than the specified minimum score. Optionally, you can specify a number to indicate that the intent must be included within that number of top elements in the array. The `top_n` parameter is ignored if you specify a negative number.

Returns `false` if the specified intent is not in the array, does not have a confidence score that is equal to or greater than the minimum confidence score, or the array index of the intent is lower than the specified index location.

The service automatically generates an `intents` array that lists the intents that the service detects in the input whenever user input is submitted. The array lists all intents that are detected by the service in order of highest confidence first.

You can use this method in a node condition to not only check for the presence of an intent, but to set a confidence score threshold that must be met before the node can be processed and its response returned.

For example, use the following expression in a node condition when you want to trigger the dialog node only when the following conditions are met:

- The `#General_Ending` intent is present.
- The confidence score of the `#General_Ending` intent is over 80%.
- The `#General_Ending` intent is one of the top 2 intents in the intents array.

```
intents.containsIntent("General_Ending", 0.8, 2)
```

JSONArray.filter(temp, "temp.property operator comparison_value")

Filters an array by comparing each array element value to a value you specify. This method is similar to a collection projection. A collection projection returns a filtered array based on a name in an array element name-value pair. The filter method returns a filtered array based on a value in an array element name-value pair.

The filter expression consists of the following values:

- `temp`: Name of a variable that is used temporarily as each array element is evaluated. For example, `city`.
- `property`: Element property that you want to compare to the `comparison_value`. Specify the property as a property of the temporary variable that you name in the first parameter. Use the syntax: `temp.property`. For example, if `latitude` is a valid element name for a name-value pair in the array, specify the property as `city.latitude`.
- `operator`: Operator to use to compare the property value to the `comparison_value`.

Supported operators are:

Operator	Description
==	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
!=	Is not equal to

Supported filter operators

- `comparison_value`: Value that you want to compare each array element property value against. To specify a value that can change depending on the user input, use a context variable or entity as the value. If you specify a value that can vary, add logic to guarantee that the `comparison_value` value is valid at evaluation time or an error occurs.

Filter example 1

For example, you can use the filter method to evaluate an array that contains a set of city names and their population numbers to return a smaller array that contains only cities with a population over 5 million.

The following `$cities` context variable contains an array of objects. Each object contains a `name` and `population` property.

```
[
  {
    "name":"Tokyo",
    "population":9273000
  },
  {
    "name":"Rome",
    "population":2868104
  },
  {
    "name":"Beijing",
    "population":20693000
  },
  {
    "name":"Paris",
    "population":2241346
  }
]
```

In the following example, the arbitrary temporary variable name is `city`. The SpEL expression filters the `$cities` array to include only cities with a population of over 5 million:

```
$cities.filter("city", "city.population > 5000000")
```

The expression returns the following filtered array:

```
[
  {
    "name":"Tokyo",
    "population":9273000
  },
  {
    "name":"Beijing",
    "population":20693000
  }
]
```

You can use a collection projection to create a new array that includes only the city names from the array that is returned by the filter method. You can then use the `join` method to display the two name element values from the array as a String, and separate the values with a comma and a space.

```
The cities with more than 5 million people include <?  T(String).join(", ",($cities.filter("city", "city.population > 5000000")).![name]) ?>.
```

The resulting response is: `The cities with more than 5 million people include Tokyo, Beijing.`

Filter example 2

The power of the filter method is that you do not need to hardcode the `comparison_value` value. In this example, the hardcoded value of 5000000 is replaced with a context variable instead.

In this example, the `$population_min` context variable contains the number `5000000`. The arbitrary temporary variable name is `city`. The SpEL expression filters the `$cities` array to include only cities with a population of over 5 million:

```
$cities.filter("city", "city.population > $population_min")
```

The expression returns the following filtered array:

```
[
  {
    "name":"Tokyo",
    "population":9273000
  },
  {
    "name":"Beijing",
    "population":20693000
  }
]
```

Tip: When you compare number values, be sure to set the context variable that is involved in the comparison to a valid value before the filter method is triggered. `Null` can be a valid value if the array element you are comparing it against might contain it. For example, if the population name and value pair for Tokyo is `"population":null`, and the comparison expression is `"city.population == $population_min"`, then `null` would be a valid value for the `$population_min` context variable.

You can use a dialog node response expression like this:

```
The cities with more than $population_min people include <?  T(String).join(", ",($cities.filter("city", "city.population > $population_min")).![name]) ?>.
```

The resulting response is: `The cities with more than 5000000 people include Tokyo, Beijing.`

Filter example 3

In this example, an entity name is used as the `comparison_value`. The user input is, `What is the population of Tokyo?`. The arbitrary temporary variable name is `y`. You created an entity that is named `@city` that recognizes city names, including `Tokyo`.

```
$ $cities.filter("y", "y.name == @city")
```

The expression returns the following array:

```
[
  {
    "name":"Tokyo",
    "population":9273000
  }
]
```

You can use a collection project to get an array with only the population element from the original array, and then use the `get` method to return the value of the population element.

```
The population of @city is: <? ($cities.filter("y", "y.name == @city").![population]).get(0) ?>.
```

The expression returns: `The population of Tokyo is 9273000.`

JSONArray.get(Integer)

This method returns the input index from the JSONArray.

For this dialog runtime context that is set by a previous node or external application:

```
{
  "context": {
    "name": "John",
    "nested": {
      "array": [ "one", "two" ]
    }
  }
}
```

Dialog node or response condition:

```
$nested.array.get(0).getAsString().contains('one')
```

Result: `True` because the nested array contains `one` as a value.

Response:

```
"output": {
  "generic" : [
    {
      "values": [
        {
        "text" : "The first item in the array is <?$nested.array.get(0)?>"
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
    }
  ]
  }
```

JSONArray.getRandomItem()

This method returns a random item from the input JSONArray.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ham"]
  }
}
```

Dialog node output:

```
{
  "output": {
  "generic" : [
    {
      "values": [
        {
    "text": "<? $toppings_array.getRandomItem() ?> is a great choice!"
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
    }
  ]
  }
}
```

Result: `"ham is a great choice!"` or `"onion is a great choice!"` or `"olives is a great choice!"`

Note: The resulting output text is randomly chosen.

JSONArray.indexOf(value)

This method returns the index number of the element in the array that matches the value you specify as a parameter or `-1` if the value is not found in the array. The value can be a String (`"School"`), Integer(`8`), or Double (`9.1`). The value must be an exact match and is case-sensitive.

For example, the following context variables contain arrays:

```
{
  "context": {
    "array1": ["Mary","Lamb","School"],
    "array2": [8,9,10],
    "array3": [8.1,9.1,10.1]
  }
}
```

The following expressions can be used to determine the array index at which the value is specified:

```
<? $array1.indexOf("Mary") ?> returns `0`
<? $array2.indexOf(9) ?> returns `1`
<? $array3.indexOf(10.1) ?> returns `2`
```

This method can be useful for getting the index of an element in an intents array, for example. You can apply the `indexOf` method to the array of intents generated each time user input is evaluated to determine the array index number of a specific intent.

```
intents.indexOf("General_Greetings")
```

If you want to know the confidence score for a specific intent, you can pass the earlier expression in as the *index* value to an expression with the syntax `intents[` *index* `].confidence` . For example:

```
intents[intents.indexOf("General_Greetings")].confidence
```

JSONArray.join(String delimiter)

This method joins all values in this array to a string. Values are converted to strings and delimited by the input delimiter.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ham"]
  }
}
```

Dialog node output:

```
{
  "output": {
  "generic" : [
    {
      "values": [
        {
    "text": "This is the array: <? $toppings_array.join(';') ?>"
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
    }
  ]
  }
}
```

Result:

```
This is the array: onion;olives;ham;
```

If a user input mentions multiple toppings, and you defined an entity that is named `@toppings` that can recognize topping mentions, you might use the following expression in the response to list the toppings that were mentioned:

```
So, you'd like <? @toppings.values.join(',') ?>.
```

If you define a variable that stores multiple values in a JSON array, you can return a subset of values from the array. Use the `join()` method to format them properly.

Collection projection

A `collection projection` SpEL expression extracts a subcollection from an array that contains objects. The syntax for a collection projection is `array_that_contains_value_sets.![value_of_interest]`.

For example, the following context variable defines a JSON array that stores flight information. Each flight has two data points, the time and flight code.

```
"flights_found": [
  {
    "time": "10:00",
    "flight_code": "OK123"
  },
  {
    "time": "12:30",
    "flight_code": "LH421"
  },
  {
    "time": "16:15",
    "flight_code": "TS4156"
  }
]
```

To return the flight codes only, you can create a collection projection expression by using the following syntax:

```
<? $flights_found.![flight_code] ?>
```

This expression returns an array of the `flight_code` values as `["OK123","LH421","TS4156"]`. See the Spring Expression Language (SpEL) documentation for more details.

If you apply the `join()` method to the values in the returned array, the flight codes are displayed in a comma-separated list. For example, you can use the following syntax in a response:

```
The flights that fit your criteria are:
  <? T(String).join(",", $flights_found.![flight_code]) ?>.
```

Result: `The flights that match your criteria are: OK123,LH421,TS4156.`

JSONArray.joinToArray(template, retainDataType)

This method extracts information from each item in the array and builds a new array that is formatted according to the template you specify. The template can be a string, a JSON object, or an array. The method returns an array of strings, an array of objects, or an array of arrays, depending on the type of template.

This method is useful for formatting information as a string that you can return as part of the output of a dialog node, or for transforming data into a different structure so you can use it with an external API.

In the template, you can reference values from the source array by using the following syntax, where `{property}` represents the name of the property in the source array.

```
%e.{property}%
```

For example, suppose that your assistant stores an array that contains flight details in a context variable. The stored data might look like this:

```
"flights": [
    {
      "flight": "AZ1040",
```

```
    "origin": "JFK",
    "carrier": "Alitalia",
    "duration": 485,
    "destination": "FCO",
    "arrival_date": "2019-02-03",
    "arrival_time": "07:00",
    "departure_date": "2019-02-02",
    "departure_time": "16:45"
  },
  {
    "flight": "DL1710",
    "origin": "JFK",
    "carrier": "Delta",
    "duration": 379,
    "destination": "LAX",
    "arrival_date": "2019-02-02",
    "arrival_time": "10:19",
    "departure_date": "2019-02-02",
    "departure_time": "07:00"
  },
  {
    "flight": "VS4379",
    "origin": "BOS",
    "carrier": "Virgin Atlantic",
    "duration": 385,
    "destination": "LHR",
    "arrival_date": "2019-02-03",
    "arrival_time": "09:05",
    "departure_date": "2019-02-02",
    "departure_time": "21:40"
  }
]
```

To build an array of strings that describe these flights in a user-readable form, you might use the following expression:

```
${Flight_data}.joinToArray("Flight %e.flight% to %e.destination%", true)
```

This expression would return the following array of strings: `["Flight AZ1040 to FCO","Flight DL1710 to LAX","Flight VS4379 to LHR"]`.

The optional `retainDataType` parameter specifies whether the method should preserve the data type of all input values in the returned array. If `retainDataType` is set to `false` or omitted, in some situations, strings in the input array might be converted to numbers in the returned array. For example, if the selected values from the input array are `"1"`, `"2"`, and `"3"`, the returned array might be `[ 1, 2, 3 ]`. To avoid unexpected type conversions, specify `true` for this parameter.

Complex templates

A more complex template might contain formatting that displays the information in a legible layout. For a complex template, you might want to store the template in a context variable, which you can then pass to the `joinToArray` method instead of a string.

For example, this complex template contains a subset of the array elements, adding labels and formatting:

```
<br/>Flight number: %e.flight% <br/> Airline: %e.carrier% <br/> Departure date: %e.departure_date% <br/> Departure time: %e.departure_time% <br/> Arrival time: %e.arrival_time% <br/>
```

Note: Make sure any formatting that you use in your template is supported by the channel integration that displays the assistant output.

If you create a context variable that is called `Template`, and assign this template as its value, you can then use that variable in your expressions:

```
${Flight_data}.joinToArray(${Template})
```

At run time, the response would look like this:

```
Flight number: AZ1040
Airline: Alitalia
Departure date: 2019-02-02
Departure time: 16:45
Arrival time: 07:00
```

```
Flight number: DL1710
Airline: Delta
Departure date: 2019-02-02
Departure time: 07:00
Arrival time: 10:19

Flight number: VS4379
Airline: Virgin Atlantic
Departure date: 2019-02-02
Departure time: 21:40
Arrival time: 09:05
```

JSON Object templates

Instead of a string, you can define a template as a JSON object. This provides a way to standardize the formatting of information from different systems, or to transform data into the format required for an external service.

In this example, a template is defined as a JSON object that extracts flight details from the elements that are specified in the array that is stored in the `Flight data` context variable:

```
{
  "departure": "Flight %e.flight% departs on %e.departure_date% at %e.departure_time%.",
  "arrival": "Flight %e.flight% arrives on %e.arrival_date% at %e.arrival_time%."
}
```

Using this template, the `joinToArray()` method returns a new array of objects with the specified structure.

JSONArray.remove(Integer)

This method removes the element in the index position from the JSONArray and returns the updated JSONArray.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives"]
  }
}
```

Make this update:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.remove(0) ?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array": ["olives"]
  }
}
```

JSONArray.removeValue(object)

This method removes the first occurrence of the value from the JSONArray and returns the updated JSONArray.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives"]
  }
}
```

Make this update:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.removeValue('onion') ?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array": ["olives"]
  }
}
```

JSONArray.set(Integer index, Object value)

This method sets the input index of the JSONArray to the input value and returns the modified JSONArray.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives", "ham"]
  }
}
```

Dialog node output:

```
{
  "context": {
    "toppings_array": "<? $toppings_array.set(1,'ketchup')?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array": ["onion", "ketchup", "ham"]
  }
}
```

JSONArray.size()

This method returns the size of the JSONArray as an integer.

For this dialog runtime context:

```
{
  "context": {
    "toppings_array": ["onion", "olives"]
  }
}
```

Make this update:

```
{
  "context": {
    "toppings_array_size": "<? $toppings_array.size() ?>"
  }
}
```

Result:

```
{
  "context": {
    "toppings_array_size": 2
  }
}
```

JSONArray split(String regexp)

This method splits the input string by using the input regular expression. The result is a JSONArray of strings.

For this input:

```
"bananas;apples;pears"
```

This syntax:

```
{
  "context": {
    "array": "<?input.text.split(";")?>
  }
}
```

Results in this output:

```
{
  "context": {
    "array": [ "bananas", "apples", "pears" ]
  }
}
```

com.google.gson.JsonArray support

In addition to the built-in methods, you can use standard methods of the `com.google.gson.JsonArray` class.

New array

`new JsonArray().append('value')`

To define a new array that is completed with values that are provided by users, you can instantiate an array. You must also add a placeholder value to the array when you instantiate it. You can use the following syntax to do so:

```
{
  "context":{
    "answer": "<? output.answer?:new JsonArray().append('temp_value') ?>"
  }
```

Date and Time

Several methods are available to work with date and time.

For information about how to recognize and extract date and time information from user input, see @sys-date and @sys-time entities.

The following string formats are supported for date-time literals on which the methods might be invoked.

- For time only: `HH:mm:ss` or `HH:mm`
- For date only: `yyyy-MM-dd`
- For date and time: `yyyy-MM-dd HH:mm:ss`
- For date and time with time zone: `yyyy-MM-dd HH:mm:ss VV`. The V symbol is from the DateTimeFormatter and represents a time zone in IANA Time Zone Database (TZDB) format, for example, Europe/London.

.after(String date or time)

Determines whether the date/time value is after the date/time argument.

.before(String date or time)

Determines whether the date/time value is before the date/time argument.

For example:

- @sys-time.before('12:00:00')
- @sys-date.before('2016-11-21')
- If you compare different items, such as `time vs. date`, `date vs. time`, and `time vs. date and time`, the method returns false and an exception is printed in the response JSON log `output.log_messages`.

  For example, `@sys-date.before(@sys-time)`.
- If you compare `date and time vs. time` the method ignores the date and compares only times.

now(String time zone)

Returns a string with the current date and time in the format `yyyy-MM-dd HH:mm:ss`. Optionally specify a `timezone` value to get the current date and time for a specific time zone, with a returned string in the format `yyyy-MM-dd HH:mm:ss 'GMT'XXX`.

- Static function.
- The other date/time methods can be invoked on date-time values that are returned by this function and it can be passed in as their argument.
- The user interface automatically creates a `$timezone` context variable for you so the correct time is returned when you test from the "Try it out" pane. If you don't pass a time zone, the time zone that is set automatically by the UI is used. Outside of the UI, `GMT` is used as the time zone. To learn about the syntax to use to specify the time zone, see Time zones that are supported by system entities.

Example of using `now()` to first check whether it's morning before the assistant responds with a morning-specific greeting.

```
{
  "conditions": "now().before('12:00:00')",
  "output": {
     "generic": [
      {
      "values": [
        {
        "text": "Good morning!"
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
      }
     ]
  }
}
```

Example of using now() with a time zone to return the current time (in England):

```
{
  "output": {
     "generic": [
      {
      "values": [
        {
        "text": "The current date and time is: <? now('Europe/London') ?>"
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
      }
     ]
  }
}
```

You can substitute the hardcoded time zone value with a context variable to dynamically change the time based on a time zone that is passed to the expression. For example: `<? now('$myzone') ?>`. The `$myzone` context variable might be set to `'Australia/Sydney'` in one conversation and to `'Mexico/BajaNorte'` in another.

.reformatDateTime(String format)

Formats date and time strings to the format you want for user output.

Returns a formatted string according to the specified format:

- `MM/dd/yyyy` for 12/31/2016
- `h a` for 10pm

To return the day of the week:

- `EEEE` for Tuesday
- `E` for Tue
- `u` for day index (1 = Monday, ..., 7 = Sunday)

For example, this context variable definition creates a $time variable that saves the value 17:30:00 as *5:30 PM*.

```
{
  "context": {
    "time": "<? @sys-time.reformatDateTime('h:mm a') ?>"
  }
}
```

The format follows the Java SimpleDateFormat rules.

Note: When you try to format time only, the date is treated as `1970-01-01`.

.sameMoment(String date/time)

- Determines whether the date/time value is the same as the date/time argument.

.sameOrAfter(String date/time)

- Determines whether the date/time value is after or the same as the date/time argument.
- Analogous to `.after()`.

.sameOrBefore(String date/time)

- Determines whether the date/time value is before or the same as the date/time argument.

today()

Returns a string with the current date in the format `yyyy-MM-dd`.

- Static function.
- The other date methods can be invoked on date values that are returned by this function and it can be passed in as their argument.
- If the context variable `$timezone` is set, this function returns dates in the client's time zone. Otherwise, the `GMT` time zone is used.

Example of a dialog node with `today()` used in the output field:

```
{
  "conditions": "#what_day_is_it",
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Today's date is <? today() ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result: `Today's date is 2018-03-09.`

Date and time calculations

Use the following methods to calculate a date, where `<date>` is specified in the format `yyyy-MM-dd` or `yyyy-MM-dd HH:mm:ss`.

Method	Description
`<date>.minusDays(n)`	Returns the date of the day n number of days before the specified date.
`<date>.minusMonths(n)`	Returns the date of the day n number of months before the specified date.
`<date>.minusYears(n)`	Returns the date of the day n number of years before the specified date.
`<date>.plusDays(n)`	Returns the date of the day n number of days after the specified date.
`<date>.plusMonths(n)`	Returns the date of the day n number of months after the specified date.
`<date>.plusYears(n)`	Returns the date of the day n number of years after the specified date.

To get tomorrow's date, specify the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Tomorrow's date is <? today().plusDays(1) ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result if today is March 9, 2018: `Tomorrow's date is 2018-03-10.`

To get the date for the day a week from today, specify the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Next week's date is <? @sys-date.plusDays(7) ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result if the date captured by the @sys-date entity is today's date, March 9, 2018: `Next week's date is 2018-03-16.`

To get last month's date, specify the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
```

```
        {
         "text": "Last month the date was <? today().minusMonths(1) ?>."
        }
       ],
       "response_type": "text",
       "selection_policy": "sequential"
     }
   ]
  }
}
```

Result if today is March 9, 2018: `Last month the date was 2018-02-9.`

Use the following methods to calculate time, where `<time>` is specified in the format `HH:mm:ss`.

Method	Description
`<time>.minusHours(n)`	Returns the time n hours before the specified time.
`<time>.minusMinutes(n)`	Returns the time n minutes before the specified time.
`<time>.minusSeconds(n)`	Returns the time n seconds before the specified time.
`<time>.plusHours(n)`	Returns the time n hours after the specified time.
`<time>.plusMinutes(n)`	Returns the time n minutes after the specified time.
`<time>.plusSeconds(n)`	Returns the time n seconds after the specified time.

To get the time an hour from now, specify the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "One hour from now is <? now().plusHours(1) ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result if it is 8 AM: `One hour from now is 09:00:00.`

To get the time 30 minutes ago, specify the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "A half hour before @sys-time is <? @sys-time.minusMinutes(30) ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result if the time captured by the @sys-time entity is 8 AM: `A half hour before 08:00:00 is 07:30:00.`

To reformat the time that is returned, you can use the following expression:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "6 hours ago was <? now().minusHours(6).reformatDateTime('h:mm a') ?>."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

Result if it is 2:19 PM: `6 hours ago was 8:19 AM.`

Working with time spans

To show a response based on whether today's date falls within a certain time frame, you can use a combination of time-related methods. For example, if you run a special offer during the holiday season every year, you can check whether today's date falls between November 25 and December 24 of this year. First, define the dates of interest as context variables.

In the following start and end date context variable expressions, the date is being constructed by concatenating the derived current year value with hardcoded month and day values.

```
$ "context": {
   "end_date": "<? now().reformatDateTime('Y') + '-12-24' ?>",
   "start_date": "<? now().reformatDateTime('Y') + '-11-25' ?>"
 }
```

In the response condition, you can indicate that you want to show the response only if the current date falls between the start and end dates that you defined as context variables.

`now().after($start_date) && now().before($end_date)`

java.util.Date support

In addition to the built-in methods, you can use standard methods of the `java.util.Date` class.

To get the date of the day that falls a week from today, you can use the following syntax.

```
{
  "context": {
    "week_from_today": "<? new Date(new Date().getTime() +
      (7 * (24*60*60*1000L))) ?>"
  }
}
```

This expression first gets the current date in milliseconds since January 1, 1970, 00:00:00 Coordinated Universal Time. It also calculates the number of milliseconds in 7 days. (The `(24*60*60*1000L)` represents one day in milliseconds.) It then adds 7 days to the current date. The result is the full date of the day that falls a week from today. For example, `Fri Jan 26 16:30:37 UTC 2018`. The time is in the Coordinated Universal Time time zone. You can always change the 7 to a variable (`$number_of_days`, for example) that you can pass in. Be sure that its value gets set before this expression is evaluated.

If you want to be able to compare the date with another date that is generated by the service, then you must reformat the date. System entities (`@sys-date`) and other built-in methods (`now()`) convert dates to the `yyyy-MM-dd` format.

```
{
  "context": {
    "week_from_today": "<? new Date(new Date().getTime() +
      (7 * (24*60*60*1000L))).format('yyyy-MM-dd') ?>"
  }
}
```

After you reformat the date, the result is `2018-01-26`. Now, you can use an expression like `@sys-date.after($week_from_today)` in a response condition to compare a date that is specified in user input to the date saved in the context variable.

The following expression calculates the time 3 hours from now.

```
{
  "context": {
    "future_time": "<? new Date(new Date().getTime() + (3 * (60*60*1000L)) -
      (5 * (60*60*1000L))).format('h:mm a') ?>"
  }
}
```

The `(60*60*1000L)` value represents an hour in milliseconds. This expression adds 3 hours to the current time. It then recalculates the time from the Coordinated Universal Time zone to the EST time zone by subtracting 5 hours from it. It also reformats the date values to include hours and minutes AM or PM.

Numbers

These methods help you get and reformat number values.

For information about system entities that can recognize and extract numbers from user input, see @sys-number entity.

If you want the service to recognize specific number formats in user input, such as order number references, consider creating a pattern entity to capture it. See Creating entities for more details.

If you want to change the decimal placement for a number, to reformat a number as a currency value, for example, see the String format() method.

toDouble()

Converts the object or field to the Double number type. You can call this method on any object or field. If the conversion fails, *null* is returned.

toInt()

Converts the object or field to the Integer number type. You can call this method on any object or field. If the conversion fails, *null* is returned.

toLong()

Converts the object or field to the Long number type. You can call this method on any object or field. If the conversion fails, *null* is returned.

If you specify a Long number type in a SpEL expression, you must append an `L` to the number to identify it as such. For example, `5000000000L`. This syntax is required for any numbers that do not fit into the 32-bit Integer type. For example, numbers that are greater than 2^31 (2,147,483,648) or less than -2^31 (-2,147,483,648) are considered Long number types. Long number types have a minimum value of -2^63 and a maximum value of 2^63-1 (or 9,223,372,036,854,775,807).

If you need to determine if a number is too long to be recognized, check if there are more than 18 integers in the number by using an expression like this:

```
<? @sys-number.toString().length() > 18 ?>
```

If you need to work with numbers that are longer than 18 integers, consider a pattern entity (with a regular expression such as `\d{20}`) to work with them instead of using `@sys-number`.

Standard math

Use SpEL expressions to define standard math equations, where the operators are represented by using these symbols:

Arithmetic operation	Symbol
addition	•
division	/

multiplication	•
subtraction	•

For example, in a dialog node response, you might add a context variable that captures a number that is specified in the user input (`@sys-number`), and saves it as `$your_number` . You can then add the following text as a text response:

```
I'm doing math. Given the value you specified ($your_number), when I add 5, I get: <? $your_number + 5 ?>.
When I subtract 5, I get: <? $your_number - 5 ?>.
When I multiply it by 5, I get: <? $your_number * 5 ?>.
When I divide it by 5, I get: <? $your_number/5 ?>.
```

If the user specifies `10` , then the resulting text response looks like this:

```
I'm doing math. Given the value you specified (10), when I add 5, I get: 15.
When I subtract 5, I get: 5.
When I multiply it by 5, I get: 50.
When I divide it by 5, I get: 2.
```

Java number support

java.lang.Math()

Performs basic numeric operations.

You can use the Class methods:

max()

```
{
  "context": {
    "bigger_number": "<? T(Math).max($number1,$number2) ?>"
  },
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "The bigger number is $bigger_number."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

min()

```
{
  "context": {
    "smaller_number": "<? T(Math).min($number1,$number2) ?>"
  },
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "The smaller number is $smaller_number."
          }
        ],
        "response_type": "text",
```

```
      "selection_policy": "sequential"
    }
   ]
  }
}
```

pow()

```
{
  "context": {
    "power_of_two": "<? T(Math).pow($base.toDouble(),2.toDouble()) ?>"
  },
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Your number $base to the second power is $power_of_two."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

For more information, see the java.lang.Math reference documentation.

java.util.Random()

Returns a random number. You can use one of the following syntax options:

- To return a random Boolean value (true or false), use `<?new Random().nextBoolean()?>`.
- To return a random double number between 0 (included) and 1 (excluded), use `<?new Random().nextDouble()?>`.
- To return a random integer between 0 (included) and a number you specify, use `<?new Random().nextInt(n)?>` where n is the top of the number range you want + 1. For example, if you want to return a random number between 0 and 10, specify `<?new Random().nextInt(11)?>`.
- To return a random integer from the full Integer value range (-2147483648 to 2147483648), use `<?new Random().nextInt()?>`.

For example, you might create a dialog node that is triggered by the #random_number intent. The first response condition might look like this:

```
Condition = @sys-number
{
  "context": {
    "answer": "<? new Random().nextInt(@sys-number.numeric_value + 1) ?>"
  },
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Here's a random number between 0 and @sys-number.literal: $answer."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  }
}
```

See the java.util.Random reference documentation for information about other methods.

You can use standard methods of the following classes also:

- `java.lang.Byte`
- `java.lang.Integer`

- `java.lang.Long`
- `java.lang.Double`
- `java.lang.Short`
- `java.lang.Float`

Objects

JSONObject.clear()

This method clears all values from the JSON object and returns null.

For example, you want to clear the current values from the $user context variable.

```
{
  "context": {
    "user": {
      "first_name":"John",
      "last_name":"Snow"
    }
  }
}
```

Use the following expression in the output to define a field that clears the object of its values.

```
{
  "output": {
    "object_eraser": "<? $user.clear() ?>"
  }
}
```

If you reference the $user context variable, it returns `{}` only.

You can use the `clear()` method on the `context` or `output` JSON objects in the body of the API `/message` call.

Clearing context

When you use the `clear()` method to clear the `context` object, it clears **all** variables except these ones:

- `context.conversation_id`
- `context.timezone`
- `context.system`

Warning: All context variable values means:

- All default values that were set for variables in nodes that were triggered during the current session.
- Any updates that are made to the default values with information that is provided by the user or external services during the current session.

To use the method, you can specify it in an expression in a variable that you define in the output object. For example:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Response for this node."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ],
    "context_eraser": "<? context.clear() ?>"
  }
}
```

Clearing output

When you use the `clear()` method to clear the `output` object, it clears all variables except the one you use to clear the output object and any text responses that you define in the current node. It also does not clear these variables:

- `output.nodes_visited`
- `output.nodes_visited_details`

To use the method, you can specify it in an expression in a variable that you define in the output object. For example:

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
          "text": "Have a great day!"
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ],
    "output_eraser": "<? output.clear() ?>"
  }
}
```

If a node earlier in the tree defines a text response of `I'm happy to help.` and then jumps to a node with the JSON output object defined earlier, then only `Have a great day.` is displayed as the response. The `I'm happy to help.` output is not displayed because it is cleared and replaced with the text response from the node that is calling the `clear()` method.

JSONObject.has(String)

This method returns true if the complex JSONObject has a property of the input name.

For this dialog runtime context:

```
{
  "context": {
    "user": {
      "first_name": "John",
      "last_name": "Snow"
    }
  }
}
```

Dialog node output:

```
{
  "conditions": "$user.has('first_name')"
}
```

Result: The condition is true because the user object contains the property `first_name`.

JSONObject.remove(String)

This method removes a property of the name from the input `JSONObject`. The `JSONElement` that is returned by this method is the `JSONElement` that is being removed.

For this dialog runtime context:

```
{
  "context": {
    "user": {
      "first_name": "John",
      "last_name": "Snow"
    }
  }
```

```
}
```

Dialog node output:

```
{
  "context": {
    "attribute_removed": "<? $user.remove('first_name') ?>"
  }
}
```

Result:

```
{
  "context": {
    "user": {
      "last_name": "Snow"
    },
    "attribute_removed": {
      "first_name": "John"
    }
  }
}
```

com.google.gson.JsonObject support

In addition to the built-in methods, some of the standard methods of the `com.google.gson.JsonObject` class are also supported.

Strings

These methods help you work with text.

For information about how to recognize and extract certain types of Strings, such as people names and locations, from user input, see System entities.

Note: For methods that involve regular expressions, see RE2 Syntax reference for details about the syntax to use when you specify the regular expression.

String.append(Object)

This method appends an input object to the string as a string and returns a modified string.

For this dialog runtime context:

```
{
  "context": {
    "my_text": "This is a text."
  }
}
```

This syntax:

```
{
  "context": {
    "my_text": "<? $my_text.append(' More text.') ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "my_text": "This is a text. More text."
  }
}
```

String.contains(String)

This method returns true if the string contains the input substring.

Input: `Yes, I'd like to go.`

This syntax:

```
{
  "conditions": "input.text.contains('Yes')"
}
```

Results: The condition is `true`.

String.endsWith(String)

This method returns true if the string ends with the input substring.

For this input:

```
"What is your name?".
```

This syntax:

```
{
  "conditions": "input.text.endsWith('?')"
}
```

Results: The condition is `true`.

String.equals(String)

This method returns `true` if the specified string equals the input string exactly.

Input: "Yes"

This syntax:

```
{
  "conditions": "input.text.equals('Yes')"
}
```

Results: The condition is `true`.

If the input is `Yes.`, then the result is `false` because the user included a period and the expression expects only the exact text, `Yes` without any punctuation.

String.equalsIgnoreCase(String)

This method returns `true` if the specified string equals the input string, regardless of whether the casing of the letters matches.

Input: "yes"

This syntax:

```
{
  "conditions": "input.text.equalsIgnoreCase('Yes')"
}
```

Results: The condition is `true`.

If the input is `Yes.`, then the result is `false` because the user included a period and the expression expects only the text `Yes`, in uppercase or lowercase letters without any punctuation.

String.extract(String regexp, Integer groupIndex)

This method returns a string from the input that matches the regular expression group pattern that you specify. It returns an empty string if no match is found.

Note: This method is designed to extract matches for different regex pattern groups, not different matches for a single regex pattern. To find

different matches, see the getMatch method.

In this example, the context variable is saving a string that matches the regex pattern group that you specify. In the expression, two regex patterns groups are defined, each one enclosed in parentheses. An inherent third group is composed of the two groups. This is the first (groupIndex 0) regex group; it matches with a string that contains the full number group and text group. The second regex group (groupIndex 1) matches with the first occurrence of a number group. The third group (groupIndex 2) matches with the first occurrence of a text group after a number group.

```
{
  "context": {
    "number_extract": "<? input.text.extract('([\\d]+)(\\b [A-Za-z]+)',n) ?>"
  }
}
```

When you specify the regex in JSON, you must provide two backslashes (\). If you specify this expression in a node response, you need one backslash only. For example:

```
<? input.text.extract('([\d]+)(\b [A-Za-z]+)',n) ?>
```

Input:

```
"Hello 123 this is 456".
```

Result:

- When n=`0`, the value is `123 this`.
- When n=`1`, the value is `123`.
- When n=`2`, the value is `this`.

String.find(String regexp)

This method returns true if any segment of the string matches the input regular expression. You can call this method against a JSONArray or JSONObject element, and it converts the array or object to a string before it makes the comparison.

For this input:

```
"Hello 123456".
```

This syntax:

```
{
  "conditions": "input.text.find('^[^\d]*[\d]{6}[^\d]*$')"
}
```

Result: The condition is true because the numeric portion of the input text matches the regular expression `^[^\d]*[\d]{6}[^\d]*$`.

String.getMatch(String regexp, Integer matchIndex)

This method returns a string from the input that matches the occurrence of the regular expression pattern that you specify. This method returns an empty string if no match is found.

As matches are found, they are added to what you can think of as a *matches array*. If you want to return the third match, because the array element count starts at 0, specify 2 as the `matchIndex` value. For example, if you enter a text string with three words that match the specified pattern, you can return the first, second, or third match only by specifying its index value.

In the following expression, you are looking for a group of numbers in the input. This expression saves the second pattern-matching string into the `$second_number` context variable because the index value 1 is specified.

```
{
  "context": {
    "second_number": "<? input.text.getMatch('([\\d]+)',1) ?>"
  }
}
```

If you specify the expression in JSON syntax, you must provide two backslashes (\). If you specify the expression in a node response, you need one backslash only.

For example:

`<? input.text.getMatch('([\d]+)',1) ?>`

- User input:

  ```
  "hello 123 i said 456 and 8910".
  ```

- Result: `456`

In this example, the expression looks for the third block of text in the input.

`<? input.text.getMatch('(\b [A-Za-z]+)',2) ?>`

For the same user input, this expression returns `and`.

String.isEmpty()

This method returns true if the string is an empty string, but not null.

For this dialog runtime context:

```
{
  "context": {
    "my_text_variable": ""
  }
}
```

This syntax:

```
{
  "conditions": "$my_text_variable.isEmpty()"
}
```

Results: The condition is `true`.

String.length()

This method returns the character length of the string.

For this input:

```
"Hello"
```

This syntax:

```
{
  "context": {
    "input_length": "<? input.text.length() ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "input_length": 5
  }
}
```

String.matches(String regexp)

This method returns true if the string matches the input regular expression.

For this input:

```
"Hello".
```

This syntax:

```
{
  "conditions": "input.text.matches('^Hello$')"
}
```

Result: The condition is true because the input text matches the regular expression `\^Hello\$`.

String.startsWith(String)

This method returns true if the string starts with the input substring.

For this input:

```
"What is your name?".
```

This syntax:

```
{
  "conditions": "input.text.startsWith('What')"
}
```

Results: The condition is `true`.

String.substring(Integer beginIndex, Integer endIndex)

This method gets a substring with the character at `beginIndex` and the last character set to be indexed before `endIndex`. The endIndex character is not included.

For this dialog runtime context:

```
{
  "context": {
    "my_text": "This is a text."
  }
}
```

This syntax:

```
{
  "context": {
    "my_text": "<? $my_text.substring(5, $my_text.length()) ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "my_text": "is a text."
  }
}
```

String.toJson()

This method parses a string that contains JSON data and returns a JSON object or array, as in this example:

```
${json_var}.toJson()
```

If the context variable `${json_var}` contains the following string:

```
"{ \"firstname\": \"John\", \"lastname\": \"Doe\" }"
```

The `toJson()` method returns the following object:

```
{
  "firstname": "John",
  "lastname": "Doe"
}
```

String.toLowerCase()

This method returns the original string that is converted to lowercase letters.

For this input:

```
"This is A DOG!"
```

This syntax:

```
{
  "context": {
    "input_lower_case": "<? input.text.toLowerCase() ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "input_lower_case": "this is a dog!"
  }
}
```

String.toUpperCase()

This method returns the original string that is converted to uppercase letters.

For this input:

```
"hi there".
```

This syntax:

```
{
  "context": {
    "input_upper_case": "<? input.text.toUpperCase() ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "input_upper_case": "HI THERE"
  }
}
```

String.trim()

This method trims any spaces at the beginning and the end of the string and returns the modified string.

For this dialog runtime context:

```
{
  "context": {
    "my_text": "   something is here    "
  }
}
```

This syntax:

```
{
  "context": {
    "my_text": "<? $my_text.trim() ?>"
  }
}
```

Results in this output:

```
{
  "context": {
    "my_text": "something is here"
  }
}
```

java.lang.String support

In addition to the built-in methods, you can use standard methods of the `java.lang.String` class.

java.lang.String.format()

You can apply the standard Java String `format()` method to text. See java.util.formatter reference for information about the syntax to use to specify the format details.

For example, the following expression takes three decimal integers (1, 1, and 2) and adds them to a sentence.

```
{
  "formatted String": "<? T(java.lang.String).format('%d + %d equals %d', 1, 1, 2) ?>"
}
```

Result: `1 + 1 equals 2`.

To change the decimal placement for a number, use the following syntax:

```
{
  <? T(String).format('%.2f',<number to format>) ?>
}
```

For example, if the `$number` variable that needs to be formatted in US dollars is `4.5`, then a response such as `Your total is $<? T(String).format('%.2f',$number) ?>` returns `Your total is $4.50.`.

Indirect data type conversion

When you include an expression within text, as part of a node response, for example, the value is rendered as a String. If you want the expression to be rendered in its original data type, then do not surround it with text.

For example, you can add this expression to a dialog node response to return the entities that are recognized in the user input in String format:

```
The entities are <? entities ?>.
```

If the user specifies *Hello now* as the input, then the @sys-date and @sys-time entities are triggered by the `now` reference. The entities object is an array, but because the expression is included in text, the entities are returned in String format, like this:

```
The entities are 2018-02-02, 14:34:56.
```

If you do not include text in the response, then an array is returned instead. For example, if the response is specified as an expression only, not surrounded by text.

```
<? entities ?>
```

The entity information is returned in its original data type, as an array.

```
[
  {
    "entity":"sys-date","location":[6,9],"value":"2018-02-02","confidence":1,"metadata":{"calendar_type":"GREGORIAN","timezone":"America/New_York"}
  },
```

```
 {
   "entity":"sys-time","location":[6,9],"value":"14:33:22","confidence":1,"metadata":{"calendar_type":"GREGORIAN","timezone":"America/New_York"}
 }
 ]
```


As another example, the following $array context variable is an array, but the $string_array context variable is a string.

```
{
  "context": {
    "array": [
      "one",
      "two"
    ],
    "array_in_string": "this is my array: $array"
  }
}
```

If you check the values of these context variables in the Try it out pane, you see their values that are specified as follows:

$array : `["one","two"]`

$array_in_string : `"this is my array: [\"one\",\"two\"]"`

You can perform array methods on the $array variable, such as `<? $array.removeValue('two') ?>`, but not the $array_in_string variable.

Expressions for accessing objects in dialog

You can write expressions that access objects and properties of objects by using the Spring Expression (SpEL) language. For more information, see Spring Expression Language (SpEL).

Evaluation syntax

To expand variable values inside other variables or invoke methods on properties and global objects, use the `<? expression ?>` expression syntax. For example:

- Expanding a property

```
"output":{"text":"Your name is <? context.userName ?>"}
```

- Invoking methods on properties of global objects

```
"context":{"email": "<? @email.literal ?>"}
```

Shorthand syntax

Learn how to quickly reference the following objects by using the SpEL shorthand syntax:

- Context variables
- Entities
- Intents

Shorthand syntax for context variables

The following table shows examples of the shorthand syntax that you can use to write context variables in condition expressions.

Shorthand syntax	Full syntax in SpEL
`$card_type`	`context['card_type']`
$(card-type)	context['card-type']
`$card_type:VISA`	`context['card_type'] == 'VISA'`
$card_type:(MASTER CARD)	context['card_type'] == 'MASTER CARD'

Shorthand syntax

You can include special characters, such as hyphens or periods, in context variable names. However, doing so can lead to problems when the SpEL expression is evaluated. The hyphen might be interpreted as a minus sign, for example. To avoid such problems, reference the variable by using either the full expression syntax or the shorthand syntax `$(variable-name)` and do not use the following special characters in the name:

- Parentheses `()`
- More than one apostrophe `''`
- Quotation marks `"`

When you refer to a context variable in a text response or a dialog node condition, you can use the short syntax.

For example, `Hello, $name`. If the `$name` context variable contains `Sam`, then the response is shown as `Hello, Sam`.

If you want to reference a context variable by using the full syntax in a text response, be sure to surround the context variable in `<? ?>`. For example, `Hello, <? context['name'] ?>`.

If you want to reference a context variable that has multiple fields, such as `$context.integrations.chat.browser_info.page_url`. To use the full syntax, specify `<? context['integrations']['chat']['browser_info']['page_url'] ?>`.

Shorthand syntax for entities

The following table shows examples of the shorthand syntax that you can use when you refer to entities.

Shorthand syntax	Full syntax in SpEL
@year	entities['year']?.value
@year == 2016	entities['year']?.value == 2016
@year != 2016	entities['year']?.value != 2016
@city == 'Boston'	entities['city']?.value == 'Boston'
@city:Boston	entities['city']?.contains('Boston')
@city:(New York)	entities['city']?.contains('New York')

Shorthand syntax

In SpEL, the question mark `(?)` prevents a null pointer exception from being triggered when an entity object is null.

If the entity value that you want to check for contains a `)` character, you cannot use the `:` operator for comparison. For example, if you want to check whether the city entity is `Dublin (Ohio)`, you must use `@city == 'Dublin (Ohio)'` instead of `@city:(Dublin (Ohio))`.

Shorthand syntax for intents

The following table shows examples of the shorthand syntax that you can use when you refer to intents.

Shorthand syntax	Full syntax in SpEL
`#help`	`intent == 'help'`
`! #help`	`intent != 'help'`
`NOT #help`	`intent != 'help'`
`#help` or `#i_am_lost`	`(intent == 'help' || intent == 'I_am_lost')`

Built-in global variables

You can use the expression language to extract property information for the following global variables:

Global variable	Definition
context	JSON object part of the processed conversation message.
entities[ ]	List of entities that supports default access to the 1st element.
input	JSON object part of the processed conversation message.
intents[ ]	List of intents that supports default access to the first element.
output	JSON object part of the processed conversation message.

Global variables

Accessing entities

The entities array contains one or more entities that were recognized in user input.

While you test your dialog, you can see details of the entities that are recognized in user input by specifying this expression in a dialog node response:

```
<? entities ?>
```

For the user input, *today*, your assistant recognizes the @sys-date system entity, so the response contains this entity object:

```
[
  {
    "entity":"sys-date",
    "location":[0,5],
    "value":"2020-12-30",
    "confidence":1.0,
```

```
    "metadata":
    {
      "calendar_type":"GREGORIAN",
      "timezone":"America/New_York"
    },
    "interpretation":
    {
      "timezone":"America/New_York",
      "relative_day":0,
      "granularity":"day",
      "calendar_type":"GREGORIAN"
     }
   }
]
```


If you want to include text in the response, use the `toJson()` method in the expression to cast the returned entities list into a JSON object. For example:

```
Recognized entities are: <? entities.toJson() ?>
```

When placement of entities in the input matters

When you use the shorthand expression `@city.contains('Boston')` in a condition, the dialog node returns true **only if** `Boston` is the first entity that is detected in the user input. Use this syntax only if the placement of entities in the input matters to you and you want to check the first mention only.

Use the full SpEL expression if you want the condition to return true anytime the term is mentioned in the user input, regardless of the order in which the entities are mentioned. The condition `entities['city']?.contains('Boston')` returns true when at least one 'Boston' city entity is found in all the @city entities, regardless of placement.

For example, a user submits `"I want to go from Toronto to Boston."` The `@city:Toronto` and `@city:Boston` entities are detected and are represented in the array that is returned as follows:

- `entities.city[0].value = 'Toronto'`
- `entities.city[1].value = 'Boston'`

Note: The order of entities in the array that is returned matches the order in which they are mentioned in the user input.

Entity properties

Each entity has a set of properties that are associated with it. You can access information about an entity through its properties.

Property	Definition	Usage tips
confidence	A decimal percentage that represents your assistant's confidence in the recognized entity. The confidence of an entity is either 0 or 1, unless you activate fuzzy matching of entities. When fuzzy matching is enabled, the default confidence level threshold is 0.3. Whether fuzzy matching is enabled, system entities always have a confidence level of 1.0.	You can use this property in a condition to have it return false if the confidence level is not higher than a percent you specify.
location	A zero-based character offsets that indicates where the detected entity values begin and end in the input text.	Use `.literal` to extract the span of text between start and end index values that are stored in the location property.
value	The entity value identified in the input.	This property returns the entity value as defined in the training data, even if the match was made against one of its associated synonyms. You can use `.values` to capture multiple occurrences of an entity that might be present in user input.

Entity properties

Entity property usage examples

In the following examples, the skill contains an airport entity that includes a value of JFK, and the synonym 'Kennedy Airport". The user input is *I want to go to Kennedy Airport*.

- To return a specific response if the 'JFK' entity is recognized in the user input, you might add this expression to the response condition: `entities.airport[0].value == 'JFK'` or `@airport = "JFK"`
- To return the entity name as it was specified by the user in the dialog response, use the `.literal` property: `So you want to go to <?entities.airport[0].literal?>...` or `So you want to go to @airport.literal ...`

Both formats evaluate to `So you want to go to Kennedy Airport...` in the response.

- Expressions like `@airport:(JFK)` or `@airport.contains('JFK')` always refer to the **value** of the entity (`JFK` in this example).
- To be more restrictive about which terms are identified as airports in the input when fuzzy matching is enabled, you can specify this expression in a node condition, for example: `@airport && @airport.confidence > 0.7`. The node executes only if your assistant is 70% confident that the input text contains an airport reference.

In this example, the user input is *Are there places to exchange currency at JFK, Logan, and O'Hare?*

- To capture multiple occurrences of an entity type in user input, use syntax like this:

```
"context":{
  "airports":"@airport.values"
}
```

  To later refer to the captured list in a dialog response, use this syntax: `You asked about these airports: <? $airports.join(', ') ?>.` It is displayed like this: `You asked about these airports: JFK, Logan, O'Hare.`
- To capture the literal values for multiple entity mentions, use the following syntax:

```
entities['myEntityName'].![literal]
```

Accessing intents

The intents array contains one or more intents that were recognized in the user input, which is sorted in descending order of confidence.

Each intent has one property only: the `confidence` property. The confidence property is a decimal percentage that represents your assistant's confidence in the recognized intent.

While you test your dialog, you can see details of the intents that are recognized in user input by specifying this expression in a dialog node response:

```
<? intents ?>
```

For the user input, *Hello now*, your assistant finds an exact match with the #greeting intent. Therefore, it lists the #greeting intent object details first. The response also includes the top 10 other intents that are defined in the skill regardless of their confidence score. (In this example, its confidence in the other intents is set to 0 because the first intent is an exact match.) The top 10 intents are returned because the "Try it out" pane sends the `alternate_intents:true` parameter with its request. If you are using the API directly and want to see the top 10 results, be sure to specify this parameter in your call. If `alternate_intents` is false, which is the default value, only intents with a confidence higher than 0.2 are returned in the array.

```
[{"intent":"greeting","confidence":1},
{"intent":"yes","confidence":0},
{"intent":"pizza-order","confidence":0}]
```

If you want to include text in the response, use the `toJson()` method in the expression to cast the returned intents list into a JSON object. For example:

```
Recognized intents are: <? intents.toJson() ?>
```

The following examples show how to check for an intent value:

- `intents[0] == 'Help'`
- `intent == 'Help'`

`intent == 'help'` differs from `intents[0] == 'help'` because `intent == 'help'` does not throw an exception if no intent is detected. It is evaluated as true only if the intent confidence exceeds a threshold. If you want to, you can specify a custom confidence level for a condition, for example, `intents.size() > 0 && intents[0] == 'help' && intents[0].confidence > 0.1`.

Accessing input

The input JSON object contains one property only: the text property. The text property represents the text of the user input.

Input property usage examples

The following example shows how to access input:

- To execute a node if the user input is "Yes", add this expression to the node condition: `input.text == 'Yes'`

You can use any of the String methods to evaluate or manipulate text from the user input. For example:

- To check whether the user input contains "Yes", use: `input.text.contains( 'Yes' )`.
- Returns true if the user input is a number: `input.text.matches( '[0-9]+' )`.
- To check whether the input string contains ten characters, use: `input.text.length() == 10`.

Phone integration context variables

You can use context variables to manage the flow of conversations with customers who interact with your assistant over the telephone.

The following tables describe context variables that have special meaning in the context of the phone integration. They need to be used for the purpose that is listed, and none other.

Context variables that are set by your dialog or actions

Name	Type	Description	Default
final_utterance_timeout_count	Number	The time (in milliseconds) that the phone integration waits to receive a final utterance from the Speech to Text service. The timeout occurs if the phone integration does not receive a final utterance within the specified time limit, even if hypotheses continue to be generated. When the timeout occurs, the phone integration sends watsonx Assistant a text update that includes the word vgwFinalUtteranceTimeout to indicate that no final utterance was received.	N/A
post_response_timeout_count	Number	The time (in milliseconds) to wait for a new utterance after the response is played back to the caller. When this timeout occurs, the phone integration channel sends a text message to the assistant that includes the word vgwPostResponseTimeout and sets the context variable input.integrations.voice_telephony.post_response_timeout_occurred to true.	7000
turn_settings.timeout_count	Number	The time (in milliseconds) to wait for a response from watsonx Assistant. If this time is exceeded, the phone integration tries again to contact watsonx Assistant. If the service still can't be reached, the call fails.	N/A
cdr_custom_data	object	Any JSON key/value pairs to collect and store with the CDR record at the end of the phone call. Each time this object is received, it is merged with any previously received cdr_custom_data context.	N/A

Voice context variables set by the dialog or actions

Example

```
{
  "generic": [
    {
      "response_type": "text",
      "text": "Hello"
    }
  ],
  "context": {
    "integrations": {
      "voice_telephony": {
        "post_response_timeout_count": 10000,
        "turn_settings": {
          "timeout_count": 5000
        },
        "cdr_custom_data": {
          "key1": "value1",
          "key2": "value2"
        }
      }
    }
  }
}
```

Context variables that are set by the phone channel

Name	Type	Description
sip_call_id	string	The SIP call ID associated with the watsonx Assistant session.

sip_custom_invite_headers	object	A JSON object with key or value pairs that define SIP headers that are pulled from the initial SIP INVITE request and passed to the watsonx Assistant service (for example, {"Custom-Header1": "123"}).
private.sip_from_uri	string	The SIP From URI associated with the watsonx Assistant service.
private.sip_request_uri	string	The SIP request URI that started the conversation session.
private.sip_to_uri	string	The SIP To URI associated with the conversation session.
private.user_phone_number	string	The phone number that the call was received from.
assistant_phone_number	string	The phone number associated with the watsonx Assistant side that received the phone call.

Context variables set by the phone channel

Input parameters that are set by the phone channel

The following input parameters are only valid for the current conversation turn.

Name	Type	Description
post_response_timeout_occurred	Boolean	Whether the post-response timeout expired
barge_in_occurred	Boolean	Whether barge-in occurred
final_utterance_timeout_occurred	true or false	Whether the final-utterance timeout expired
dtmf_collection_succeeded	Boolean	Whether the DTMF collection succeeded or failed. When true, a DTMF collection succeeded, and returns the expected number of digits. When false, a DTMF collection failed to collect the specified number of digits. Even when dtmf_collection_succeeded is false, all collected digits are passed to the dialog in the input string of the turn request.
is_dtmf	Boolean	Whether the input to watsonx Assistant is dual-tone multi-frequency signaling (DTMF).
speech_to_text_result	object	The final response from the Speech to Text service in JSON format, including the transcript and confidence score for the lead hypothesis and any alternatives. The format matches exactly the format that is received from the Speech to Text service. (For more information, see the Speech to Text API documentation.)
sms_message	string	An SMS message received from the caller

Input parameters set by the phone channel

Example

```
{
  "input": {
    "text": "agent ",
    "integrations": {
      "voice_telephony": {
        "speech_to_text_result": {
          "result_index": 0,
          "stopTimestamp": "2021-09-29T17:43:31.036Z",
          "transaction_ids": {
            "x-global-transaction-id": "43dd6ce0-139a-4d76-95aa-86e03fcfc434",
            "x-dp-watson-tran-id": "6e60695e-fed7-4efe-a376-0888b027d30f"
          },
          "results": [
            {
              "final": true,
              "alternatives": [
```

```
            {
              "transcript": "agent ",
              "confidence": 0.78
            }
          ]
        }
      ],
      "transactionID": "43dd6ce0-139a-4d76-95aa-86e03fcfc434",
      "startTimestamp": "2021-09-29T17:43:29.436Z"
    },
    "is_dtmf": false,
    "barge_in_occurred": false
  }
 }
},
"context": {
 "skills": {
  "main skill": {
   "user_defined": {},
   "system": {}
  }
 },
 "integrations": {
  "voice_telephony": {
   "private": {
    "sip_to_uri": "sip:watson-conversation@10.10.10.10",
    "sip_from_uri": "sip:10.10.10.11",
    "sip_request_uri": "sip:test@10.10.10.10:5064;transport=tcp"
   },
   "sip_call_id": "QjryZsuAS4",
   "assistant_phone_number": "18882346789"
  }
 }
}
}
```

SMS integration reference

Add action commands to the message `context` object to manage the flow of conversations with customers who interact with your assistant by submitting SMS messages over the telephone.

Learn about the supported commands and reserved context variables that are used by the *SMS* integration.

Supported commands

Each action consists of a `command` property, followed by an optional `parameter` property to define parameters for commands that require them. The commands that are described in the following table are supported by the *SMS* integration.

Action command	Description	Parameters
`terminateSession`	Ends the current SMS session. Use this command to ensure that the subsequent text message starts a new assistant-level session which does not retain any context values from the current session.	None
`smsActSendMedia`	Enables MMS messaging.	`mediaURL`: Specifies a JSON array of publicly accessible media URLs that are sent to the user.
`smsActSetDisambiguationConfig`	Configures how to handle the choices that are displayed in a disambiguation list.	`prefixText`: Text to include before each option. For example, `Press %s for` where `%s` represents the number corresponding to a list choice; this is replaced with the actual number at run time.
`smsActSetOptionsConfig`	Configures how to handle option response types.	`prefixText`: Text to include before each option. For example, `Press %s for` where `%s` represents the number corresponding to a list choice; this is replaced with the actual number at run time.

Actions that you can initiate from the action

Reserved context variables

The following table describes the context variables that have special meaning in the context of the *SMS* integration. They should not be used for any purpose other than the documented use.

Table 2 describes the context variables that are set by your action. Table 3 describes the context variables that you can set by the *SMS* integration.

Table 2. Context variables that are set by your action

Context variable name	Expected value	Description
`smsConversationResponseTimeout`	Time in ms	The amount of time in milliseconds that the integration waits to receive a response from the action. If the time limit is exceeded, the integration attempts to contact the action again. If the service still can't be reached, the SMS response fails.

SMS context variables set by the action

Table 3. Context variables that are set by the integration

Context variable name	Description
`smsTenantPhoneNumber`	The integration tenant phone number that the user is messaging.
`smsUserPhoneNumber`	The phone number of the user that is exchanging messages with the integration.

`smsUserData`	Data in JSON format to be passed verbatim to the service orchestration engine or watsonx Assistant service. This variable is sent only if the session is started from the integration tenant and the data is sent through the REST API.
`smsSessionTimeoutCount`	The session timeout value. This variable is sent only if the timeout value is defined through the REST API.
`smsError`	When the integration fails to send an SMS message, this variable contains details about the error that occurred.
`smsSessionID`	The globally unique identifier (GUID) for the related SMS Gateway session.
`smsMedia`	The `arraylist` of `mediaURL` and corresponding `mediaContentType`. This context variable is cleared at the end of each conversation turn.

SMS context variables set by the integration

CDR log event reference

For `cdr_logged` events sent by a log webhook, the `payload` object contains data about a call detail record (CDR) event that was handled by the phone integration. The `payload` object for a CDR event contains the following properties.

Property	Type	Description
`primary_phone_number`	string	Phone number that was called.
`global_session_id`	string	Unique session identifier.
`failure_occurred`	boolean	Whether a failure occurred during the call.
`failure_details`	string	Details about any failure that occurred.
`transfer_occurred`	boolean	Whether an attempt was made to transfer a call.
`active_calls`	Number	Number of active calls when the call started.
`x-global-sip-trunk-call-id`	string	Value of the SIP trunk call ID header extracted from the initial SIP `INVITE` request. The following SIP trunk call ID headers are supported: • `X-Twilio-CallSid` • `X-SID` • `X-Global-SIP-Trunk-Call-ID`
`call`	Object	Information about the call. See `call`.
`session_initiation_protocol`	Object	SIP protocol-related details. See `session_initiation_protocol`.
`max_response_milliseconds`	Object	Maximum latency for services used during the call. See `max_response_milliseconds`.
`assistant_interaction_summaries`	Array	Details about the watsonx Assistant interactions that took place during the call. See `assistant_interaction_summaries`.
`injected_custom_data`	Object	A set of key/value pairs extracted from the `cdr_custom_data` context variable.
`warnings_and_errors`	Array	Warnings or errors that were logged during the call. See `warnings_and_errors`.
`realtime_transport_network_summary`	Object	Statistics for the inbound stream in the `inbound_stream` object and statistics for the outbound stream in the `outbound_stream` object. Included only if RTCP is enabled. See `realtime_transport_network_summary`.

Properties of the CDR webhook payload object

call

The `call` object contains the following properties.

Property	Type	Description
`start_timestamp`	String	Time when the call started, in ISO format (`yyyy-MM-ddTHH:mm:ss.SSSZ`).
`stop_timestamp`	String	Time when the call ended, in ISO format (`yyyy-MM-ddTHH:mm:ss.SSSZ`).
`milliseconds_elapsed`	Number	Duration of the call, in milliseconds.

end_reason	string	Reason the call ended. Possible reasons are: • assistant_transfer • assistant_hangup • caller_hangup • failed
security.media_encrypted	Boolean	Whether the media was encrypted.
security.signaling_encrypted	Boolean	Whether the SIP signaling was encrypted.
security.sip_authenticated	Boolean	Whether SIP authentication was used to authenticate the caller.

Properties of the call object

session_initiation_protocol

The `session_initiation_protocol` object contains the following properties.

Property	Type	Description
invite_arrival_timestamp	String	Time when the `INVITE` request arrived, in ISO format (`yyyy-MM-ddTHH:mm:ss.SSSZ`).
setup_milliseconds	Number	Time that it took to set up the call, in milliseconds. The time between when the initial SIP `INVITE` request was received and when the final SIP `ACK` request was received.
headers.call_id	String	SIP `Call-ID` header field pulled from the SIP `INVITE` related to the call.
headers.from_uri	String	SIP URI from the initial SIP INVITE `From` header.
headers.to_uri	String	SIP URI from the initial SIP INVITE `To` header.

Properties of the session_initiation_protocol object

assistant_interaction_summaries

The `assistant_interaction_summaries` object contains the following properties.

Property	Type	Description
assistant_id	String	Unique identifier of the assistant.
session_id	String	Unique identifier of the session.
turns	Array	An array of objects describing the watsonx Assistant interactions that took place during the conversation. See assistant_interaction_summaries.turns[].

Properties of the assistant_interaction_summaries object

assistant_interaction_summaries.turns[]

Objects in the `assistant_interaction_summaries.turns` array contain the following properties.

Property	Type	Description
assistant.log_id	String	Unique identifier for the logged event, which can be used to correlate between message logs and CDR events.

assistant.start_timestamp	String	Time when the request was sent to the assistant, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
assistant.response_milliseconds	Number	Time (in milliseconds) between when the request was sent and when the response was received from the assistant.
request	Object	A request sent to the assistant. See assistant_interaction_summaries.turns[].request.
response	Array	An array of the response objects associated with the request.

Properties of the objects in the assistant_interaction_summaries.turns[] array

assistant_interaction_summaries.turns[].request

The assistant_interaction_summaries.turns[].request object contains the following properties.

Property	Type	Description
type	String	Request type: • start : an initial request is sent to the assistant • speech_to_text : input is received from the Speech to Text service • dtmf : DTMF collection completes • sms : an SMS message is received from the caller • post_response_timeout : the post-response timer expires • redirect : a call is redirected • transfer : a call is transferred • transfer_failed : a call transfer fails • final_utterance_timeout : the final utterance timer expires • no_input_turn : no input turn is enabled • sms_failure : an SMS message cannot be sent to the caller • speech_to_text_result_filtered : an utterance is filtered due to a low confidence level • mrcp_recognition_unsuccessful : the MRCP recognition completes without a final utterance • network_warning : a network error is detected • media_capability_change : media capabilities change during a call.
streaming_statistics	Object	Information and statistics that are related to the Speech to Text recognition. See assistant_interaction_summaries.turns[].request.streaming_statistics.

Properties of the assistant_interaction_summaries.turns[].request object

assistant_interaction_summaries.turns[].request.streaming_statistics

The assistant_interaction_summaries.turns[].request.streaming_statistics object contains the following properties.

Property	Type	Description
transaction_id	String	Unique identifier of the transaction.
start_timestamp	String	Time when the transaction started, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
stop_timestamp	String	Time when the transaction ended, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
response_milliseconds	Number	The latency (in milliseconds) between when silence is detected in the caller's speech and a final result from the assistant is received.
echo_detected	Boolean	Whether an echo was detected.

confidence	Number	Confidence score of the final utterance.

Properties of the assistant_interaction_summaries.turns[].streaming_statistics object

assistant_interaction_summaries.turns[].response

The `assistant_interaction_summaries.turns[].response` object contains the following properties.

Property	Type	Description
type	String	Response type: • `text_to_speech`: a command to play an utterance to the caller • `sms`: a command to send an SMS message to the caller • `url`: a command to play an audio file to the caller • `transfer`: a command to transfer a call • `text_to_speech_config`: a command to change the Text to Speech settings • `speech_to_text_config`: a command to change the Speech to Text settings • `pause_speech_to_text`: a command to stop speech recognition • `unpause_speech_to_text`: a command to start speech recognition • `pause_dtmf`: a command to stop processing of inbound DTMF signals • `unpause_dtmf`: unpause_dtmf: a command to start processing of inbound DTMF signals • `enable_speech_barge_in`: a command to enable speech barge-in so that callers can interrupt playback by speaking • `disable_speech_barge_in`: a command to disable speech barge-in so that playback isn't interrupted when callers speak during audio playback • `enable_dtmf_barge_in`: a command to enable DTMF barge-in so that callers can interrupt playback from the phone integration by pressing a key • `disable_dtmf_barge_in`: a command to disable DTMF barge-in so that playback from the phone integration isn't interrupted when the caller presses a key • `dtmf`: a command to send DTMF signals to the caller • `hangup`: a command to disconnect the call See Mapping between CDR and watsonx Assistant response types.
barge_in_occurred	Boolean	Whether barge-in occurred during the turn.
streaming_statistics	Object	Information and statistics that are related to Text to Speech synthesis and playback. See assistant_interaction_summaries.turns[].response.streaming_statistics.

Properties of the assistant_interaction_summaries.turns[].response object

Mapping between CDR and watsonx Assistant response types

The values of the `type` property map to watsonx Assistant response types.

CDR response type	watsonx Assistant response type
text_to_speech	text
url	audio
dtmf	dtmf, command_info.type : send
sms	user_defined, vgwAction.command : vgwActSendSMS
transfer	connect_to_agent

text_to_speech_config	text_to_speech, command_info.type : configure
speech_to_text_config	speech_to_text, command_info.type : configure
unpause_speech_to_text	start_activities, type:speech_to_text_recognition
pause_speech_to_text	stop_activities, type:speech_to_text_recognition
unpause_dtmf	start_activities, type:dtmf_collection
pause_dtmf	stop_activities, type:dtmf_collection
enable_speech_barge_in	text_to_speech, command_info.type : enable_barge_in
disable_speech_barge_in	text_to_speech, command_info.type : disable_barge_in
enable_dtmf_barge_in	dtmf, command_info.type : enable_barge_in
disable_dtmf_barge_in	dtmf, command_info.type : disable_barge_in
hangup	end_session

assistant_interaction_summaries.turns[].response.streaming_statistics

The `assistant_interaction_summaries.turns[].response.streaming_statistics` object contains the following properties.

Property	Type	Description
transaction_id	String	Unique identifier of the transaction.
start_timestamp	String	Time when the transaction started, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
stop_timestamp	String	Time when the transaction ended, in ISO format (yyyy-MM-ddTHH:mm:ss.SSSZ).
response_milliseconds	Number	Time (in milliseconds) between when a text utterance is sent to the assistant and when the phone integration receives the first packet of synthesized audio.

Properties of the assistant_interaction_summaries.turns[].response.streaming_statistics object

warnings_and_errors

The `warnings_and_errors` object contains warnings and errors that were logged during the call, which are listed in order of occurrence. Warnings for the following conditions are included.

- Messages when utterances are filtered out by the confidence score threshold.
- Text to Speech underflows, which is when Text to Speech synthesis can't keep up with the phone integration streaming rate and audio might skip.
- RTP network warnings, such as high packet loss or high average jitter, if RTCP is enabled.

The following example shows the structure of the `warnings_and_errors` object:

```
$  "warnings_and_errors": [
   {
    "message": "CWSMR0032W: A Watson Speech to Text final utterance has a confidence score of 0.1, which does not meet the confidence score threshold of 0.2. The utterance will be ignored.",
    "id": "CWSMR0032W"
   },
   {
    "message": "CWSMR0031W: The synthesis stream from the Watson Text-to-Speech service can't keep up with the playback rate to the caller, so audio might skip. transaction ID=a1b2c3d4e5",
    "id": "CWSMR0031W"
```

```
  }
]
```

The object for each warning contains the following properties.

Property	Type	Description
message	String	Text of the warning message.
id	String	Unique message identifier.

Properties of the warnings_and_errors object

max_response_milliseconds

The `max_response_milliseconds` object contains the following properties.

Property	Type	Description
assistant	Number	Maximum round-trip latency (in milliseconds), calculated from all watsonx Assistant requests related to the call.
text_to_speech	Number	Maximum time (in milliseconds) between when a text utterance is sent to the Text to Speech service and when the phone integration receives the first packet of synthesized audio. This value is calculated from all Text to Speech requests related to the call.
speech_to_text	Number	Maximum latency (in milliseconds) between when silence is detected in the caller's speech and when a final result from the Speech to Text service is received. This value is calculated from all Speech to Text recognition results related to the call.

Properties of the max_response_milliseconds object

realtime_transport_network_summary

When RTCP is enabled, the `realtime_transport_network_summary` object provides statistics for the inbound stream in the `inbound_stream` object and statistics for the outbound stream in the `outbound_stream` object.

The following example shows the structure of the `realtime_transport_network_summary` object.

```
$ "realtime_transport_network_summary": {
  "inbound_stream": {
    "maximum_jitter": 5,
    "average_jitter": 1,
    "packets_lost": 0,
    "packets_transmitted": 1000,
    "canonical_name": "user@example.com",
    "tool_name": "User SIP Phone"
   },
  "outbound_stream": {
    "maximum_jitter": 5,
    "average_jitter": 1,
    "packets_lost": 0,
    "packets_transmitted": 2000,
    "canonical_name": "voice.gateway@127.0.0.1",
    "tool_name": "IBM Voice Gateway/1.0.0.5"
   }
}
```

The object for each stream contains the following properties.

Properties	Type	Description
maximum_jitter	Number	Maximum jitter during the call.
average_jitter	Number	Average jitter calculated over the duration of the call.

packets_lost	Number	An estimate of the number of packets that were lost during the call.
packets_transmitted	Number	Estimate of the total number of packets that were transmitted during the call.
canonical_name	String	Unique identifier for the sender of the stream, typically in @ format.
tool_name	String	Name of the application or tool where the stream originated. For the phone integration, the default is IBM Voice Gateway/ .

Properties of the realtime_transport_network_summary object

Segment event reference

The following tables show details of the events that watsonx Assistant sends to Segment using the Segment extension. These events appear as tables in your Segment warehouse, and as regular events in other Destinations.

Note: Only events generated using the watsonx Assistant v2 API and associated with a user ID are included.

Message Handled

Sent when the assistant completes handling of a message.

Property	Type	Description
`accountId`	String	The ID of the IBM account.
`assistantId`	String	The ID of the assistant.
`browser`	String	The browser that was used to send the message.
`channel`	String	The channel the customer used to send the message (for example, `phone` or `chat`).
`device`	String	The type of device that was used to send the message.
`environment`	String	The environment in which the message was handled (such as `draft` or `live`.)
`language`	String	The language of the assistant.
`pageUrl`	String	The URL of the web page from which the message was sent.
`serviceInstance`	String	The IBM watsonx Assistant service instance.
`sessionId`	String	The ID of the session during which the message was handled.
`skillsInvoked`	String[]	An array of strings listing all skills that were invoked during handling of the message (for example, `main skill` or `actions skill`).

The following properties are included only for messages that were handled by an actions skill:

Property	Type	Description
`action`	String	The unique identifier of the action that was visited during handling of the message (for example, `action_202`).
`actionCompleted`	Boolean	Whether the action completed during handling of the message.
`actionCompletedReason`	String	The reason the action completed (for example, `all_steps_done` or `fallback`.)
`actionStarted`	Boolean	Whether processing of the action started during handling of the message.
`actionTitle`	String	The title of the action that was visited during handling of the message (for example, `I want to pay my bill`).
`actionsVisited`	String[]	An array of strings listing the actions visited during handling of the message.
`fallbackReason`	String	The reason why the fallback action was visited (for example, escalated to human agent or no action matches).
`handler`	String	The name of any handler that was called.

stepsVisited	Object[]	An object with two properties, stepId and stepTitle, which lists the steps that the user visited during handling of the message.
subaction	String	The name of any other action that was called by the action that was triggered by the message.

The following properties are included only for messages that were handled by a dialog skill:

Property	Type	Description
branchExited	Boolean	Whether the dialog branch was exited during handling of the message.
branchExitedReason	String	The reason the dialog branch was exited (for example, completed).
nodesVisited	String[]	An array of strings listing the dialog nodes visited during handling of the message. For each dialog node, the string specifies the node title (if any) or the node ID.

Action Started

Sent when processing of an action begins.

Property	Type	Description
accountId	String	The ID of the IBM account.
action	String	The unique identifier of the action (for example, action_202).
actionTitle	String	The title of the action (for example, I want to pay my bill).
actionCompleted	Boolean	Whether the action completed during the same conversation turn.
actionCompletedReason	String	The reason the action completed (for example, all_steps_done or fallback.)
assistantId	String	The ID of the assistant.
browser	String	The browser that was used to send the message that triggered the action.
channel	String	The channel the customer used to send the message that triggered the action (for example, phone or chat).
device	String	The type of device that was used to send the message that triggered the action.
environment	String	The environment in which the action was started (such as draft or live.)
fallbackReason	String	The reason why the fallback action started (for example, escalated to human agent or no action matches).
handler	String	The name of any handler that was called.
language	String	The language of the assistant.
pageUrl	String	The URL of the web page from which the message that triggered the action was sent.
serviceInstance	String	The IBM watsonx Assistant service instance.
sessionId	String	The ID of the session during which the message that started the action was sent.
skillsInvoked	String[]	An array of strings listing all skills that were invoked during handling of the message that started the action (for example, main skill or actions skill).

stepsVisited	Object[]	An object with two properties, stepId and stepTitle, which lists the steps that the user visited during handling of the message.
subaction	String	The name of any other action that the action called during processing.

Action Completed

Sent when processing of an action ends.

Property	Type	Description
accountId	String	The ID of the IBM account.
action	String	The unique identifier of the action (for example, action_202).
actionCompletedReason	String	The reason the action completed (for example, all_steps_done or fallback.)
actionStarted	Boolean	Whether the action was started during the same conversation turn.
actionTitle	String	The title of the action (for example, I want to pay my bill).
assistantId	String	The ID of the assistant.
browser	String	The browser that was used to send the message that triggered the action.
channel	String	The channel the customer used to send the message that started the action (for example, phone or chat).
device	String	The type of device that was used to send the message that triggered the action.
environment	String	The environment in which the action completed (such as draft or live.)
fallbackReason	String	The reason why the fallback action was called (for example, escalated to human agent or no action matches).
handler	String	The name of any handler that was called by the action.
language	String	The language of the assistant.
pageUrl	String	The URL of the web page from which the message that triggered the action was sent.
serviceInstance	String	The IBM watsonx Assistant service instance.
sessionId	String	The ID of the session during which the message that started the action was sent.
skillsInvoked	String[]	An array of strings listing all skills that were invoked during handling of the message that started the action (for example, main skill or actions skill).
stepsVisited	Object[]	An object with two properties, stepId and stepTitle, which lists the steps that the user visited during handling of the message.
subaction	String	The name of any other action that the action called during processing.

Session Started

Sent when a new session is started.

Note: The v2 stateless API does not generate events for starting sessions.

Property	Type	Description
`accountId`	String	The ID of the IBM account.
`assistantId`	String	The ID of the assistant.
`browser`	String	The browser that was used to send the message that started the session.
`channel`	String	The channel that started the session (for example, `phone` or `chat`).
`device`	String	The type of device that was used to send the message that started the session.
`environment`	String	The environment in which the session was started (such as `draft` or `live`.)
`pageUrl`	String	The URL of the web page from which the message that started the session was sent.
`serviceInstance`	String	The IBM watsonx Assistant service instance.
`sessionId`	String	The ID of the session.

Filter query reference

The watsonx Assistant service REST API offers powerful log search capabilities through filter queries. You can use the v2 `/logs` API `filter` parameter to search your skill log for events that match a specified query.

The `filter` parameter is a cacheable query that limits the results to items that match the specified filter. You can filter on various objects that are part of the JSON response model (for example, the user input text, the detected intents and entities, or the confidence score).

To see examples of filter queries, see Examples.

For more information about the `/logs` `GET` method and its response model, refer to the API Reference.

Filter query syntax

The following example shows the general form of a filter query:

Location	Query operator	Term
`request.input.text`	`::`	`"IBM Watson"`

- The *location* identifies the field that you want to filter on (in this example `request.input.text`).
- The *query operator*, which specifies the type of matching you want to use (fuzzy matching or exact matching).
- The *term* specifies the expression or value that you want to use to evaluate the field for matching. The term can contain literal text and operators, as described in the next section.

Filtering by intent or entity requires a slightly different syntax from filtering on other fields. For more information, see Filtering by intent or entity.

Note: The filter query syntax uses some characters that are not allowed in HTTP queries. Make sure that all special characters, including spaces and quotation marks, are URL encoded when sent as part of an HTTP query. For example, the filter `response_timestamp<2020-01-01` would be specified as `response_timestamp%3C2020-01-01`.

Operators

You can use the following operators in your filter query.

Operator	Description
`:`	Fuzzy match query operator. Prefix the query term with `:` if you want to match any value that contains the query term, or a grammatical variant of the query term. Fuzzy matching is available for user input text and entity values.
`::`	Exact match query operator. Prefix the query term with `::` if you want to match only values that exactly equal the query term.
`:!`	Negative fuzzy match query operator. Prefix the query term with `:!` if you want to match only values that do *not* contain the query term or a grammatical variant of the query term.
`::!`	Negative exact match query operator. Prefix the query term with `::!` if you want to match only values that do *not* exactly match the query term.
`<=`, `>=`, `>`, `<`	Comparison operators. Prefix the query term with these operators to match based on arithmetic comparison.
`` ` ` ``	Escape operator. Use in queries that include control characters that would otherwise be parsed as operators. For example, `!hello` would match the text `!hello`.
`""`	Literal phrase. Use to enclose a query term that contains spaces or other special characters. No special characters within the quotation marks are parsed by the API.
`~`	Approximate match. Append this operator followed by a `1` or `2` to the end of the query term to specify the allowed number of single-character differences between the query string and a match in the response object. For example, `car~1` would match `car`, `cat`, or `cars`, but it would not match `cats`. This operator is not valid when you filter on `log_id` or any date or time field, or with fuzzy matching.

*	Wildcard operator. Matches any sequence of zero or more characters. This operator is not valid when you filter on `log_id`, `language`, `request.context.system.assistant_id`, `workspace_id`, `request.context.metadata.deployment`, or any date or time field.
(), []	Grouping operators. Use to enclose a logical grouping of multiple expressions by using Boolean operators.
|	Boolean *or* operator.
,	Boolean *and* operator.

Filtering by intent or entity

Because of differences in how intents and entities are stored internally, the syntax for filtering on a specific intent or entity is different from the syntax that is used for other fields in the returned JSON. To specify an `intent` or `entity` field within an `intents` or `entities` collection, you must use the `:` match operator instead of a dot.

For example, this query matches any logged event where the response includes a detected intent named `hello`:

```
response.output.intents:intent::hello
```

Note the `:` operator in place of a dot (intents:intent)

Use the same pattern to match on any field of a detected intent or entity in the response. For example, this query matches any logged event where the response includes a detected entity with the value `soda`:

```
response.output.entities:value::soda
```

Similarly, you can filter on intents or entities that are sent as part of the request, as in this example:

```
request.input.intents:intent::hello
```

Filtering by other fields

To filter on another field in the log data, specify the location as a path that identifies the levels of nested objects in the JSON response from the /logs API. Use dots (`.`) to specify successive levels of nesting in the JSON data. For example, the location `request.input.text` identifies the user input text field as shown in the following JSON fragment:

```
$  "request": {
   "input": {
     "text": "Good morning"
   }
 }
```

Filtering is not available for all fields. You can filter on the following fields:

- `assistant_id`
- `customer_id`
- `language`
- `request.context.global.system.user_id`
- `request.input.text`
- `request_timestamp`
- `response.context.global.system.user_id`
- `response.output.entities`
- `response.output.intents`
- `response_timestamp`
- `session_id`
- `skill_id`
- `snapshot`

Filtering on other fields is not currently supported.

Examples

The following examples illustrate various types of queries by using this syntax.

Description	Query
The date of the response is in the month of July 2020.	`response_timestamp>=2020-07-01,response_timestamp<2020-08-01`
The timestamp of the response is earlier than `2019-11-01T04:00:00.000Z`.	`response_timestamp<2019-11-01T04:00:00.000Z`
The message is labeled with the customer ID `my_id`.	`customer_id::my_id`
The message was sent to a specific assistant.	`assistant_id::dcd5c5ad-f3a1-4345-89c5-708b0b5ff4f7`
The user input text contains the word "order" or a grammatical variant (for example, `orders` or `ordering`.	`request.input.text:order`
An intent name in the response exactly matches `place_order`.	`response.output.intents:intent::place_order`
An entity name in the response exactly matches `beverage`.	`response.output.entities:entity::beverage`
No intent name in the response exactly matches `order`.	`response.intents:intent::!order`
The user input text does not contain the word "order" or a grammatical variant.	`request.input.text:!order`
The user input text contains the string `!hello`.	`request.input.text:!hello`
The user input text contains the string `IBM Watson`.	`request.input.text:"IBM Watson"`
The user input text contains a string that has no more than 2 single-character differences from `Watson`.	`request.input.text:Watson~2`
The user input text contains a string that consists of `comm`, followed by zero or more characters, followed by `s`.	`request.input.text:comm*s`
An intent name in the response exactly matches either `hello` or `goodbye`.	`response.output.intents:intent::(hello|goodbye)`
An intent name in the response exactly matches `order`, and an entity name in the response exactly matches `beverage`.	`[response.output.intents:intent::order,response.output.entities:entity::beverage]`

Filtering v1 logs

If your application is still using the v1 API, you can query and filter logs by using the v1 /logs method. The filtering syntax is the same, but the structure of v1 logs and message requests is different. For more information, see API Reference.

With the v1 /logs API, you can filter on the following fields:

- `language`
- `meta.message.entities_count`
- `request.context.metadata.deployment`
- `request.context.system.assistant_id`
- `request.input.text`
- `response.context.conversation_id`
- `response.entities`
- `response.input.text`
- `response.intents`
- `response.top_intent`
- `workspace_id`

Filtering on other fields is not currently supported.

System entities

Learn about system entities that are provided by IBM for you to use out of the box. These built-in utility entities help your assistant recognize terms and references that are commonly used by customers in conversation, such as numbers and dates.

For information about how to add system entities to your dialog, see Creating entities.

@sys-currency entity

The `@sys-currency` system entity detects mentions of monetary currency values in user input. The currency can be expressed with a currency symbol or currency-specific terms. In either case, a number is returned.

Recognized formats

- 20 cents
- Five dollars
- $10

Attributes

- `.literal`: Exact phrase in input that is interpreted to be a currency mention.
- `.numeric_value`: Canonical numeric value as an integer or a double, in base units.
- `.location`: Lists the index element values of the first and last letters in the text string for the phrase that is interpreted to be a currency mention.
- `.unit`: Base unit of currency that is specified as a 3-letter ISO currency code. For example, `USD` or `EUR`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example (input = `twenty dollars`)	Example (input = `$1,234.56`)
@sys-currency	string	`20`	`1234.56`
@sys-currency.literal	string	twenty dollars	$1,234.56
@sys-currency.numeric_value	number	`20`	`1234.56`
@sys-currency.location	array	[0,14]	[0,9]
@sys-currency.unit	string	`USD`	`USD`

@sys-currency examples

For the input `veinte euro` (`twenty euro` in Spanish) or `€1.234,56`, @sys-currency returns these values:

Attribute	Type	Input is `veinte euro`	Input is `€1.234,56`
@sys-currency	string	20	1234.56
@sys-currency.literal	string	veinte euro	€1.234,56
@sys-currency.numeric_value	number	20	1234.56
@sys-currency.location	array	[0,11]	[0,9]
@sys-currency.unit	string	EUR	EUR

More @sys-currency examples

Equivalent results are returned for other supported languages and national currencies.

@sys-currency usage tips

If you use the `@sys-currency` entity as a node condition and the user specifies `$0` as the value, the value is recognized as a currency properly, but the condition is evaluated to the number zero, not the currency zero. As a result, the `null` in the condition is evaluated to false and the node is not processed. To check for currency values in a way that handles zeros properly, use the expression `@sys-currency OR @sys-currency == 0` in the node condition instead.

For more information about currencies that are recognized per language, see Currency support.

@sys-date

The `@sys-date` system entity detects mentions of dates in user input. The date value is stored as a string in the format `yyyy-MM-dd`. For example, the mention `May 8` is stored as `"2020-05-08"`. The system augments missing elements of a date (such as the year for `"May 8"`) with the current date values.

If you enter an invalid date in the user input, the `@sys-date` system entity does not recognize the date and shows and `Invalid date` error.

If you select English as the language, the system uses US English (`en-us`) as the locale. For the US English locale only, the format of the date is `MM/DD/YYYY`. The format of the date changes to `DD/MM/YYYY` only if the first two numbers are greater than 12. The value that is stored has the format `yyyy-MM-dd`.

Recognized formats

- Friday
- today
- May 8

Attributes

- `.alternatives`: Saves alternative date values when the input is open to interpretation. Typically the future date is returned as the `@sys-date` value. If there is ambiguity, the past date is saved as an alternative date. For example, for the input `in January`, a future date is returned. However, if it's February (January passed for this year), then the past January is returned also as an alternative date. An alternative date is also provided for weekdays when the weekday is specified without a modifier. For example, for the input `Are you open Wednesday?`, the future date is stored, but an alternative date with the date of the previous Wednesday is created also. Alternative dates are created for phrases that contain modifiers such as `next`, `this`, or `last`, which can have different meanings in different locales. The watsonx Assistant service treats `last` and `next` dates as references to the most immediate last or next day that is referenced, which might be in either the same or a previous week. For some modifiers, an alternative date is created or not depending on the current day of the week. For example, maybe the input contains `this Wednesday`. If it's Monday, the future date is stored and no alternative date is created. However, if it's Wednesday, then the date is assumed to be the Wednesday of the coming week, but today's date is stored as an alternative date. Each alternative date is saved as an object that contains a value and confidence for each alternative date. The alternative date objects are stored in a JSON Array. To get an alternative value, use the syntax: `@sys-date.alternative`.
- `.calendar_type`: Specifies the calendar type for the time zone in use. `GREGORIAN` is the only supported type.
- `.datetime_link`: See Date and time mentions.
- `.day`: Helper method that returns the day in the date.
- `.day_of_week`: Returns the day of the week in the date as a lowercase string.
- `.festival`: Recognizes locale-specific holiday names and can return the date of the upcoming holiday, such as `Thanksgiving`, `Christmas`, and `Halloween`. Use all lowercase letters if you need to specify a holiday name, in a condition for example (`@sys-date.festival == 'thanksgiving'`). For more information about recognized holidays, see Festivals.
- `.granularity`: Recognizes time frame mentions. Options are `day`, `weekend`, `week`, `fortnight`, `month`, `quarter`, and `year`.
- `.interpretation`: The object that is returned by your assistant, which contains new fields that increase the precision of the system entity classifier. You can omit `interpretation` from the expression that you use to refer to its fields. For example, you can access the value of the `festival` field in the interpretation object by using the shorthand syntax `<? @sys-date.festival ?>`.
- `.literal`: Exact phrase in input that is interpreted to be the date.
- `.location`: Index element values of the first and last letters in the text string for the phrase that is interpreted to be a date.
- `.month`: Helper method that returns the month in the date.
- `.range_link`: If present, indicates that the user's input contains a date range. The start and end index values of the string that includes the date range are saved as part of the link name. Additional information is provided, including the role that each `@sys-date` plays in the range relationship. For example, the start date has a role type of `date_from` and the end date has a role type of `date_to`. To check a role value, you can use the syntax `@sys-date.role?.type == 'date_from'`. If the user input implies a range, but only one date is specified, then the `range_link` property is not returned, but one role type is returned. For example, if the user asks `Have you been open since yesterday`. Yesterday's date is recognized as the `@sys-date` mention, and a role of type `date_from` is returned for it. A `range_modifier` that identifies the word in the input that triggers the identification of a range is also returned. In this example, the modifier is `since`.
- `.relative_{timeframe}`: Recognizes and captures mentions of relative date values in the user's input. Returns a number that shows the number of units between now and the specified date. The {time frame} units can be `year`, `month`, `week`, `weekend`, or `day`.
- `.specific_{timeframe}`: Recognizes and captures mentions of specific date units in the user's input. The {time frame} units can be `year`, `quarter`, `month`, `day`, or `day_of_week`.

- `.year`: Helper method that returns the year in the date.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-date	string	If the input is `May 13, 2020`, returns `2020-05-13`.
@sys-date.alternatives	array	For example, if today's date is `10 March 2019`, and the user enters `on March 1`, then the March 1st date for next year is saved as the `@sys-date` (`2020-03-01`). However, the user might mean March 1st of 2019. The system also saves this year's date (`2019-03-01`) as an alternative date value. As another example, if today is Tuesday and a user enters, `this Tuesday` or `next Tuesday`, then `@sys-date` is set to the date of Tuesday of next week and no alternative dates are created.
@sys-date.calendar_type	string	For any date mention, returns `GREGORIAN`.
@sys-date.datetime_link	string	N/A
@sys-date.day	string	If the input is `13 May 2020`, returns `13`.
@sys-date.day_of_week	string	If the input is `13 May 2020` and it's a Wednesday, returns `wednesday`.
@sys-date.festival	string	If the input contains `Thanksgiving Day`, then `2020-11-26` is returned.
@sys-date.granularity	string	If the input mentions `next year`, returns `year`.
@sys-date.literal	string	If the input is, `I plan to leave on Saturday.` , then `on Saturday` is returned.
@sys-date.location	array	If the input is `My vacation starts tomorrow.`, returns `[19, 27]`.
@sys-date.month	string	If the input is `13 May 2020`, returns `5`.
@sys-date.range_link	string	For example, the input might be `I will travel from March 15 to March 22` . Each `@sys-date` entity that is detected has a `range_link` property in its output that is named `date_range_14_39` where the location of the date range text is `[14,39]`. The start date (`2020-05-15`) has a role type of `date_from`. The end date (`2020-05-22`) has a role type of `date_to`.
@sys-date.relative_{timeframe}	number	If the input is `two weeks ago`, returns `relative_week = -2`. If the input is `this weekend`, returns `relative_weekend = 0`. If the input is `today`, returns `relative_day = 0`.
@sys-date.specific_{timeframe}	string	For `2020`, returns `specific_year = 2020`. For `Q2 2020` or `second quarter 2020`, returns `specific_quarter = 2`. For `March`, returns `specific_month = 3`. For `Monday`, returns `specific_day_of_week = monday`. For `May 3rd`, returns `specific_day = 3`.
@sys-date.year	string	If the input is `13 May 2020`, returns `2020`.

@sys-date attributes

@sys-time

The `@sys-time` system entity detects mentions of times in user input. Returns the time that is specified in user input in the format `HH:mm:ss`. For example, `"13:00:00"` for `1pm`.

Recognized formats

- 2pm
- at 4
- 15:30

Attributes

- `.alternatives`: Captures times other than the one saved as the `@sys-time` value that the user might mean when the time is not clearly indicated. The alternative values are saved as an object that contains a value and confidence for each alternative time and is stored in a JSON Array. To get an alternative value, use the syntax: `@sys-time.alternative`.
- `.calendar_type`: Specifies the calendar type for the time zone in use. `GREGORIAN` is the only supported type.
- `.granularity`: Recognizes mentions of time frames. Options are `hour`, `minute`, `second`, and `instant`.
- `.hour`: Helper method that returns the hour that is specified in the time as a numeric value.
- `.interpretation`: The object that is returned by your assistant, which contains new fields that increase the precision of the system entity classifier. You can omit `interpretation` from the expression that you use to refer to its fields. For example, you can access the value of the `granularity` field in the interpretation object by using the shorthand syntax `<? @sys-time.granularity ?>`.
- `.literal`: Exact phrase in the input that is interpreted to be the time.
- `.location`: Lists the index element values of the first and last letters in the text string for the phrase that is interpreted to be a time mention.
- `.minute`: Helper method that returns the minute value that is specified in the time.
- `.part_of_day`: Recognizes terms that represent the time of day, such as `morning`, `afternoon`, `evening`, `night`, or `now`. Also sets a time range for each part of the day. The response contains two `@sys-time` values, one for the start and one for the end of the time range. The entities array contains a `range_link` object with `time-from` and `time-to` role types. The time ranges for different parts of the day can differ by locale. For US English, morning is `6:00:00` to `12:00:00`. Afternoon is `12:00:00` to `18:00:00`. Evening is `18:00:00` to `22:00:00`. Night is `22:00:00` to `23:59:59`. Night ends just before midnight because otherwise it would overlap with a new day, which is measured by `@sys-date`.
- `.range_link`: If present, indicates that the user's input contains a time range. The start and end index values of the string that includes the time range are saved as part of the link name. Additional information is provided, including the role that each `@sys-time` plays in the range relationship. For example, the start time has a role type of `time_from` and the end time has a role type of `time_to`. To check a role value, you can use the syntax `@sys-time.role?.type == 'time_from'`. If the user input implies a range, but only one time is specified, then the `range_link` property is not returned, but one role type is returned. A `range_modifier` that identifies the word in the input that triggers the identification of a range is also returned.
- `.relative_{timeframe}`: Recognizes relative mentions of time and returns a number that shows the number of units between now and the specified time. The {timeframe} units can be `hour`, `minute`, or `second`.
- `.second`: Helper method that returns the second value that is specified in the time.
- `.specific_{timeframe}`: Recognizes and captures mentions of specific time units in the user input. The {timeframe} units can be `hour`, `minute`, or `second`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-time	string	For the input `6:30 PM`, returns `18:30:00`.
@sys-time.alternatives	array	If a user says `The meeting is at 5`, the system calculates the time as `5:00:00` or `5 AM` and saves that as the `@sys-time`. However, the user might mean `17:00:00` or `5 PM`. The system also saves the later time as an alternative time value. If the user enters `The meeting is at 5pm`, then `@sys-time` is set to `17:00:00`, and no alternative value is created because there is no AM and PM confusion.
@sys-time.calendar_type	string	For any time mention, returns `GREGORIAN`.
@sys-time.granularity	string	If the input is `now`, returns `instant`. For `3 o'clock` or `noon`, returns `hour`. For `17:00:00`, returns `second`.
@sys-time.hour	number	If the input contains `5:30:10 PM`, it returns `17` to represent 5 PM.
@sys-time.literal	string	For the input, `The store closes at 8PM`, returns `at 8PM`.
@sys-time.location	array	For the input, `The store closes at 8PM`, returns `[17, 23]`.
@sys-time.minute	number	If the input mentions the time `5:30:10 PM`, returns `30`. If no minutes are specified, then this helper returns `0`.
@sys-time.part_of_day	string	If the input is `this morning`, returns two @sys-time values, `6:00:00` and `12:00:00`. Also returns a `range_link` object with `time-from` and `time-to` role types.
@sys-time.range_link	string	For the input `Are you open from 9AM to 11AM`, two `@sys-time` entities are detected. Each `@sys-time` entity that is detected has a `range_link` property in its output. If a range is implied but only one time value is provided, then only one `@sys-time` value is returned. For the input `Are you open until 9PM`, 9PM is recognized as the `@sys-time` mention, and a role of type `time_to` is returned for it. The `range_modifier` in this example is `until`.

@sys-time.relative_{timeframe}	number	For `5 hours ago`, returns `relative_hour = -5`. For `in two minutes`, returns `relative_minute = 2`. For `in a second`, returns `relative_second = 1`.
@sys-time.second	number	If the input mentions the time `5:30:10 PM`, returns `10`. If no second value is specified, then this helper returns `0`.
@sys-time.specific_{timeframe}	string	For `at 5 o'clock`, returns `specific_hour = 5`. For `at 2:30`, returns `specific_minute = 30`. For `23:30:22`, returns `specific_second = 22`.

@sys-time attributes

Date and time mentions

Some user input contains information that includes both date and time information. Your assistant captures both values and saves them as two separate entity mentions, one `@sys-date` and one `@sys-time`.

The relationship between the date and time values is established in two ways:

- Each entity that participates in the relationship contains the same `datetime_link` attribute.
- The literal string that spans the complete date and time mention, when they are mentioned together in input, is the same for both entities. The location information is appended to the `datetime_link` name.

Recognized formats

- now
- two hours from now
- on Monday at 4pm

Attributes

- `.datetime_link`: If present, this indicates that the user's input mentions a date and time together, which implies that the date and time are related to one another. The start and end index values of the string that includes the date and time are saved as part of the link name.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
`@sys-date.datetime_link` and `@sys-time.datetime_link`	string	If the input is, `Are you open today at 5?`, then `today` is recognized as a date mention (`2020-05-21`) at location `[13,18]` and `at 5` is recognized as a time mention (`05:00:00`) at location `[19,23]`. A location value of `[13,23]`, which spans both the date and time mentions, is appended to the resulting `datetime_link`. It is named `datetime_link_13_23` and it is included in the output of the `@sys-date` and `@sys-time` entities that participate in the relationship.

Date and time attributes

Time zones

Mentions of a date or time that are relative to the current time are resolved for a chosen time zone. By default, the time zone is Greenwich mean time. Therefore, REST API clients that are located in different time zones get the current Coordinated Universal Time when `now` is mentioned in input.

Optionally, the REST API client can add the local time zone as the context variable `$timezone`. This context variable must be sent with every client request. For example, the `$timezone` value can be `America/Los_Angeles`, `EST`, or `UTC`. For a full list of supported time zones, see Supported time zones.

When the `$timezone` variable is provided, the values of relative `@sys-date` and `@sys-time` mentions are computed based on the client time zone instead of Coordinated Universal Time.

Tip: For information about processing date and time values, see the Date and time method reference.

@sys-number entity

The `@sys-number` system entity detects mentions of numbers in user input. The number can be written with either numerals or words. In either case, a number is returned.

Recognized formats

- 21
- twenty one
- 3.13

Attributes

- `.literal`: Exact phrase in input that is interpreted to be the number.
- `.location`: Index element values of the first and last letters in the text string that is interpreted to be a number.
- `.numeric_value`: Canonical numeric value as an integer or a double.
- `.range_link`: If present, indicates that the user's input contains a number range. The location value of the text that specifies the range is appended to the link name. Additional information is provided, including the role that each `@sys-number` plays in the range relationship. For example, the start number has a role type of `number_from` and the end number has a role type of `number_to`. To check a role value, you can use the syntax `@sys-number.role?.type == 'number_from'`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-number	string	If the input is `twenty`, returns `20`. If the input is `1,234.56`, returns `1234.56`.
@sys-number.literal	string	If the input is `twenty`, returns `twenty`. If the input is `1,234.56` returns `1,234.56`.
@sys-number.location	array	If the input is `twenty`, returns [0,6]. If the input is `1,234.56`, returns [0,8]
@sys-number.numeric_value	number	If the input is `twenty`, returns `20`. If the input is `1,234.56`, returns `1234.56`.
@sys-number.range_link	string	If the input is `I'm interested in 5 to 7.`. Each of the two `@sys-number` system entities that are detected have a `range_link` of `number_range_18_24` in their output. The first `@sys-number` entity (`5`) has a `number_from` role type. The last `@sys-number` entity (`7`) has a `number_to` role type.

@sys-number examples

@sys-number usage tips

- If you use the @sys-number entity as a node condition and the user specifies zero as the value, the 0 value is recognized properly as a number. However, the 0 is interpreted as a `null` value for the condition, which results in the node not being processed. To check for numbers in a way that handles zeros properly, use the expression `@sys-number == 0` in the node condition also. The full expression to use is `@sys-number OR @sys-number == 0`.
- If you use @sys-number to compare number values in a condition, be sure to separately include a check for the presence of a number itself. If no number is found, @sys-number evaluates to null. Your comparison might evaluate to true even when no number is present.

  For example, do not use `@sys-number<4` alone because if no number is found, `@sys-number` evaluates to null. Because null is less than 4, the condition evaluates to true even though no number is present.

  Use `(@sys-number OR @sys-number == 0) AND @sys-number < 4` instead. If no number is present, the first condition evaluates to false. As a result, the whole condition evaluates to false.

Tip: For more information about processing number values, see the Numbers method reference.

@sys-percentage entity

The `@sys-percentage` system entity detects mentions of percentages in user input. The percentage can be expressed in an utterance with the percent symbol or written out using the word `percent`. In either case, a numeric value is returned.

Recognized formats

- 15%
- 10 percent

Attributes

- `.literal`: Exact phrase in input that is interpreted to be a percentage.
- `.location`: Index element values of the first and last letters in the text string that is interpreted to be a percentage mention.
- `.numeric_value`: Canonical numeric value as an integer or a double
- `.range_link`: If present, indicates that the user's input contains a range, such as `from 2 to 3 percent`. The location value of the text that specifies the range is appended to the link name. Additional information is provided, including the role that each `@sys-percentage` plays in the range relationship. For example, the start number has a role type of `number_from` and the end number has a role type of `number_to`. To check a role value, you can use the syntax `@sys-percentage.role?.type == 'number_from'`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-percentage	string	If input contains `1,234.56%`, returns `1234.56`.
@sys-percentage.literal	string	If input contains `50 percent` returns `50 percent`.
@sys-percentage.location	array	If input contains `from 2 to 3 percent`, returns `[0,19]`.
@sys-percentage.numeric_value	number	If input contains `50 percent` returns `50`.
@sys-percentage.range_link	string	If input contains `from 2 to 3 percent`, returns `number_range_0_19`.

@sys-percentage examples

@sys-percentage usage tips

- If you use the @sys-percentage entity as a node condition and the user specifies `0%` as the value, the value is recognized as a percentage properly, but the condition is evaluated to the number zero not the percentage 0%. As a result, the `null` in the condition is evaluated to false and the node is not processed. To check for percentages in a way that handles zero percentages properly, use the expression `@sys-percentage OR @sys-percentage == 0` in the node condition instead.
- If you input a value like `1-2%`, the values `1%` and `2%` are returned as system entities.

Using system entities in conditions

In a node that conditions on the `#Customer_Care_Store_Hours` intent, you can add conditional responses that use new system entity properties to provide slightly different answers about store hours depending on what the user asks.

Tip: You probably do not want to use all of these conditional responses in a real dialog; they are described here merely to illustrate what's possible.

Conditional response condition syntax	Description	Example response text
`@sys-date.festival == 'thanksgiving'`	Checks whether the customer is asking about hours on Thanksgiving day in particular.	We give out free meals to those in need on Thanksgiving Day.
`@sys-date.festival`	Checks whether the customer is asking about hours on another specific holiday.	We are closed on Christmas Day, July 4th, and President's Day. On other holidays, we are open from 10AM to 5PM.
`@sys-time.part_of_day == 'night'`	Checks whether the user includes any terms that mention night as the time of day in the input.	We are not open late; we close at 9PM most days.

`@sys-date.datetime_link && input.text.contains('today') && now().sameOrAfter(@sys-time)`	Checks whether the input contains a phrase such as `today at 8`. It also checks whether the `@sys-time` that is detected is earlier than the current time of day. You can add a context variable to the conditional response that creates a `$new_time` variable with the value `<? @sys-time.plusHours(12) ?>`. You can use this approach to create a date variable that captures the more likely time that is meant by a user who, at noon time asks, `Are you open today at 8`. The system assumes that the user means 8AM instead of 8PM.	You mean today at $new_time, correct?
`@sys-time.range_link && entities[0].role?.type == 'time_from' && entities[1].role?.type == 'time_to'`	Checks whether the input contains a time range. The condition also makes sure that the first `@sys-time` system entity that is listed in the entities array is the start time and that the second one is the end time before including them in the response.	You want to know if we're open between `<? entities[0] ?>` and `<? entities[1] ?>`, is that correct?
`@sys-time && entities.size() < 2 && entities[0].role?.type == 'time_to'`	Checks whether the input contains one `@sys-time` mention that has a `time_to` role type. If the user input matches this condition, it suggests that the user specified the end time of an open-ended time range. For example, the response would be triggered by the input `Are you open until 9pm?`. This input matches because it identifies only one time in a time span, and the mention has a `time_to` role.	Do you mean from now (`<? now().reformatDateTime('h:mm a') ?>`) until `<? @sys-time.reformatDateTime('h:mm a') ?>`?
`@sys-date.day_of_week == 'sunday'`	Checks whether a specific date that the user is asking about falls on a Sunday.	We are closed on Sundays.
`@sys-date.specific_day_of_week == 'monday' && @sys-date.alternative`	Checks whether the user mentioned the weekday `Monday` in their query. For example, `Are you open Monday`. The condition also checks whether any alternative dates were stored. Alternative dates are created when your assistant isn't entirely sure which Monday the user means, so it stores the dates of alternative Mondays also. When a user specifies a weekday, your assistant assumes that the user means the future occurrence of the day (the coming Monday). You can add a response that double checks the intended date, by using the detected alternative value. In this case, the alternative date is the previous Monday's date.	Do you mean `@sys-date` or `@sys-date.alternative`?
`true`	Responds to any other requests for store hour information.	We're open from 9AM to 9PM Monday through Saturday.

System entities in conditional responses

System entities

Learn about system entities that are provided by IBM for you to use out of the box. These built-in utility entities help your assistant recognize terms and references that are commonly used by customers in conversation, such as numbers and dates.

For information about how to add system entities to your dialog, see Creating entities.

@sys-currency entity

The `@sys-currency` system entity detects mentions of monetary currency values in user input. The currency can be expressed with a currency symbol or currency-specific terms. In either case, a number is returned.

Recognized formats

- 20 cents
- Five dollars
- $10

Attributes

- `.literal`: Exact phrase in input that is interpreted to be a currency mention.
- `.numeric_value`: Canonical numeric value as an integer or a double, in base units.
- `.location`: Lists the index element values of the first and last letters in the text string for the phrase that is interpreted to be a currency mention.
- `.unit`: Base unit of currency that is specified as a 3-letter ISO currency code. For example, `USD` or `EUR`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example (input = `twenty dollars`)	Example (input = `$1,234.56`)
@sys-currency	string	`20`	`1234.56`
@sys-currency.literal	string	twenty dollars	$1,234.56
@sys-currency.numeric_value	number	`20`	`1234.56`
@sys-currency.location	array	[0,14]	[0,9]
@sys-currency.unit	string	`USD`	`USD`

@sys-currency examples

For the input `veinte euro` (`twenty euro` in Spanish) or `€1.234,56`, @sys-currency returns these values:

Attribute	Type	Input is `veinte euro`	Input is `€1.234,56`
@sys-currency	string	20	1234.56
@sys-currency.literal	string	veinte euro	€1.234,56
@sys-currency.numeric_value	number	20	1234.56
@sys-currency.location	array	[0,11]	[0,9]
@sys-currency.unit	string	EUR	EUR

More @sys-currency examples

Equivalent results are returned for other supported languages and national currencies.

@sys-currency usage tips

If you use the `@sys-currency` entity as a node condition and the user specifies `$0` as the value, the value is recognized as a currency properly, but the condition is evaluated to the number zero, not the currency zero. As a result, the `null` in the condition is evaluated to false and the node is not processed. To check for currency values in a way that handles zeros properly, use the expression `@sys-currency OR @sys-currency == 0` in the node condition instead.

For more information about currencies that are recognized per language, see Currency support.

@sys-date

The `@sys-date` system entity detects mentions of dates in user input. The date value is stored as a string in the format `yyyy-MM-dd`. For example, the mention `May 8` is stored as `"2020-05-08"`. The system augments missing elements of a date (such as the year for `"May 8"`) with the current date values.

If you enter an invalid date in the user input, the `@sys-date` system entity does not recognize the date and shows and `Invalid date` error.

If you select English as the language, the system uses US English (`en-us`) as the locale. For the US English locale only, the format of the date is `MM/DD/YYYY`. The format of the date changes to `DD/MM/YYYY` only if the first two numbers are greater than 12. The value that is stored has the format `yyyy-MM-dd`.

Recognized formats

- Friday
- today
- May 8

Attributes

- `.alternatives`: Saves alternative date values when the input is open to interpretation. Typically the future date is returned as the `@sys-date` value. If there is ambiguity, the past date is saved as an alternative date. For example, for the input `in January`, a future date is returned. However, if it's February (January passed for this year), then the past January is returned also as an alternative date. An alternative date is also provided for weekdays when the weekday is specified without a modifier. For example, for the input `Are you open Wednesday?`, the future date is stored, but an

alternative date with the date of the previous Wednesday is created also. Alternative dates are created for phrases that contain modifiers such as `next`, `this`, or `last`, which can have different meanings in different locales. The watsonx Assistant service treats `last` and `next` dates as references to the most immediate last or next day that is referenced, which might be in either the same or a previous week. For some modifiers, an alternative date is created or not depending on the current day of the week. For example, maybe the input contains `this Wednesday`. If it's Monday, the future date is stored and no alternative date is created. However, if it's Wednesday, then the date is assumed to be the Wednesday of the coming week, but today's date is stored as an alternative date. Each alternative date is saved as an object that contains a value and confidence for each alternative date. The alternative date objects are stored in a JSON Array. To get an alternative value, use the syntax: `@sys-date.alternative`.

- `.calendar_type`: Specifies the calendar type for the time zone in use. `GREGORIAN` is the only supported type.
- `.datetime_link`: See Date and time mentions.
- `.day`: Helper method that returns the day in the date.
- `.day_of_week`: Returns the day of the week in the date as a lowercase string.
- `.festival`: Recognizes locale-specific holiday names and can return the date of the upcoming holiday, such as `Thanksgiving`, `Christmas`, and `Halloween`. Use all lowercase letters if you need to specify a holiday name, in a condition for example (`@sys-date.festival == 'thanksgiving'`). For more information about recognized holidays, see Festivals.
- `.granularity`: Recognizes time frame mentions. Options are `day`, `weekend`, `week`, `fortnight`, `month`, `quarter`, and `year`.
- `.interpretation`: The object that is returned by your assistant, which contains new fields that increase the precision of the system entity classifier. You can omit `interpretation` from the expression that you use to refer to its fields. For example, you can access the value of the `festival` field in the interpretation object by using the shorthand syntax `<? @sys-date.festival ?>`.
- `.literal`: Exact phrase in input that is interpreted to be the date.
- `.location`: Index element values of the first and last letters in the text string for the phrase that is interpreted to be a date.
- `.month`: Helper method that returns the month in the date.
- `.range_link`: If present, indicates that the user's input contains a date range. The start and end index values of the string that includes the date range are saved as part of the link name. Additional information is provided, including the role that each `@sys-date` plays in the range relationship. For example, the start date has a role type of `date_from` and the end date has a role type of `date_to`. To check a role value, you can use the syntax `@sys-date.role?.type == 'date_from'`. If the user input implies a range, but only one date is specified, then the `range_link` property is not returned, but one role type is returned. For example, if the user asks `Have you been open since yesterday`. Yesterday's date is recognized as the `@sys-date` mention, and a role of type `date_from` is returned for it. A `range_modifier` that identifies the word in the input that triggers the identification of a range is also returned. In this example, the modifier is `since`.
- `.relative_{timeframe}`: Recognizes and captures mentions of relative date values in the user's input. Returns a number that shows the number of units between now and the specified date. The {time frame} units can be `year`, `month`, `week`, `weekend`, or `day`.
- `.specific_{timeframe}`: Recognizes and captures mentions of specific date units in the user's input. The {time frame} units can be `year`, `quarter`, `month`, `day`, or `day_of_week`.
- `.year`: Helper method that returns the year in the date.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-date	string	If the input is `May 13, 2020`, returns `2020-05-13`.
@sys-date.alternatives	array	For example, if today's date is `10 March 2019`, and the user enters `on March 1`, then the March 1st date for next year is saved as the `@sys-date` (`2020-03-01`). However, the user might mean March 1st of 2019. The system also saves this year's date (`2019-03-01`) as an alternative date value. As another example, if today is Tuesday and a user enters, `this Tuesday` or `next Tuesday`, then `@sys-date` is set to the date of Tuesday of next week and no alternative dates are created.
@sys-date.calendar_type	string	For any date mention, returns `GREGORIAN`.
@sys-date.datetime_link	string	N/A
@sys-date.day	string	If the input is `13 May 2020`, returns `13`.
@sys-date.day_of_week	string	If the input is `13 May 2020` and it's a Wednesday, returns `wednesday`.
@sys-date.festival	string	If the input contains `Thanksgiving Day`, then `2020-11-26` is returned.
@sys-date.granularity	string	If the input mentions `next year`, returns `year`.

@sys-date.literal	string	If the input is, `I plan to leave on Saturday.` , then `on Saturday` is returned.
@sys-date.location	array	If the input is `My vacation starts tomorrow.`, returns `[19, 27]`.
@sys-date.month	string	If the input is `13 May 2020`, returns `5`.
@sys-date.range_link	string	For example, the input might be `I will travel from March 15 to March 22` . Each `@sys-date` entity that is detected has a `range_link` property in its output that is named `date_range_14_39` where the location of the date range text is `[14,39]`. The start date (`2020-05-15`) has a role type of `date_from`. The end date (`2020-05-22`) has a role type of `date_to`.
@sys-date.relative_{timeframe}	number	If the input is `two weeks ago`, returns `relative_week = -2`. If the input is `this weekend`, returns `relative_weekend = 0`. If the input is `today`, returns `relative_day = 0`.
@sys-date.specific_{timeframe}	string	For `2020`, returns `specific_year = 2020`. For `Q2 2020` or `second quarter 2020`, returns `specific_quarter = 2`. For `March`, returns `specific_month = 3`. For `Monday`, returns `specific_day_of_week = monday`. For `May 3rd`, returns `specific_day = 3`.
@sys-date.year	string	If the input is `13 May 2020`, returns `2020`.

@sys-date attributes

@sys-time

The `@sys-time` system entity detects mentions of times in user input. Returns the time that is specified in user input in the format `HH:mm:ss`. For example, `"13:00:00"` for `1pm`.

Recognized formats

- 2pm
- at 4
- 15:30

Attributes

- `.alternatives`: Captures times other than the one saved as the `@sys-time` value that the user might mean when the time is not clearly indicated. The alternative values are saved as an object that contains a value and confidence for each alternative time and is stored in a JSON Array. To get an alternative value, use the syntax: `@sys-time.alternative`.
- `.calendar_type`: Specifies the calendar type for the time zone in use. `GREGORIAN` is the only supported type.
- `.granularity`: Recognizes mentions of time frames. Options are `hour`, `minute`, `second`, and `instant`.
- `.hour`: Helper method that returns the hour that is specified in the time as a numeric value.
- `.interpretation`: The object that is returned by your assistant, which contains new fields that increase the precision of the system entity classifier. You can omit `interpretation` from the expression that you use to refer to its fields. For example, you can access the value of the `granularity` field in the interpretation object by using the shorthand syntax `<? @sys-time.granularity ?>`.
- `.literal`: Exact phrase in the input that is interpreted to be the time.
- `.location`: Lists the index element values of the first and last letters in the text string for the phrase that is interpreted to be a time mention.
- `.minute`: Helper method that returns the minute value that is specified in the time.
- `.part_of_day`: Recognizes terms that represent the time of day, such as `morning`, `afternoon`, `evening`, `night`, or `now`. Also sets a time range for each part of the day. The response contains two `@sys-time` values, one for the start and one for the end of the time range. The entities array contains a `range_link` object with `time-from` and `time-to` role types. The time ranges for different parts of the day can differ by locale. For US English, morning is `6:00:00` to `12:00:00`. Afternoon is `12:00:00` to `18:00:00`. Evening is `18:00:00` to `22:00:00`. Night is `22:00:00` to `23:59:59`. Night ends just before midnight because otherwise it would overlap with a new day, which is measured by `@sys-date`.
- `.range_link`: If present, indicates that the user's input contains a time range. The start and end index values of the string that includes the time range are saved as part of the link name. Additional information is provided, including the role that each `@sys-time` plays in the range relationship. For example, the start time has a role type of `time_from` and the end time has a role type of `time_to`. To check a role value, you can use the syntax `@sys-time.role?.type == 'time_from'`. If the user input implies a range, but only one time is specified, then the `range_link` property is not returned, but one role type is returned. A `range_modifier` that identifies the word in the input that triggers the identification of a range is also returned.
- `.relative_{timeframe}`: Recognizes relative mentions of time and returns a number that shows the number of units between now and the specified time. The {timeframe} units can be `hour`, `minute`, or `second`.

- `.second`: Helper method that returns the second value that is specified in the time.
- `.specific_{timeframe}`: Recognizes and captures mentions of specific time units in the user input. The {timeframe} units can be `hour`, `minute`, or `second`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-time	string	For the input `6:30 PM`, returns `18:30:00`.
@sys-time.alternatives	array	If a user says `The meeting is at 5`, the system calculates the time as `5:00:00` or `5 AM` and saves that as the `@sys-time`. However, the user might mean `17:00:00` or `5 PM`. The system also saves the later time as an alternative time value. If the user enters `The meeting is at 5pm`, then `@sys-time` is set to `17:00:00`, and no alternative value is created because there is no AM and PM confusion.
@sys-time.calendar_type	string	For any time mention, returns `GREGORIAN`.
@sys-time.granularity	string	If the input is `now`, returns `instant`. For `3 o'clock` or `noon`, returns `hour`. For `17:00:00`, returns `second`.
@sys-time.hour	number	If the input contains `5:30:10 PM`, it returns `17` to represent 5 PM.
@sys-time.literal	string	For the input, `The store closes at 8PM`, returns `at 8PM`.
@sys-time.location	array	For the input, `The store closes at 8PM`, returns `[17, 23]`.
@sys-time.minute	number	If the input mentions the time `5:30:10 PM`, returns `30`. If no minutes are specified, then this helper returns `0`.
@sys-time.part_of_day	string	If the input is `this morning`, returns two @sys-time values, `6:00:00` and `12:00:00`. Also returns a `range_link` object with `time-from` and `time-to` role types.
@sys-time.range_link	string	For the input `Are you open from 9AM to 11AM`, two `@sys-time` entities are detected. Each `@sys-time` entity that is detected has a `range_link` property in its output. If a range is implied but only one time value is provided, then only one `@sys-time` value is returned. For the input `Are you open until 9PM`, 9PM is recognized as the `@sys-time` mention, and a role of type `time_to` is returned for it. The `range_modifier` in this example is `until`.
@sys-time.relative_{timeframe}	number	For `5 hours ago`, returns `relative_hour = -5`. For `in two minutes`, returns `relative_minute = 2`. For `in a second`, returns `relative_second = 1`.
@sys-time.second	number	If the input mentions the time `5:30:10 PM`, returns `10`. If no second value is specified, then this helper returns `0`.
@sys-time.specific_{timeframe}	string	For `at 5 o'clock`, returns `specific_hour = 5`. For `at 2:30`, returns `specific_minute = 30`. For `23:30:22`, returns `specific_second = 22`.

@sys-time attributes

Date and time mentions

Some user input contains information that includes both date and time information. Your assistant captures both values and saves them as two separate entity mentions, one `@sys-date` and one `@sys-time`.

The relationship between the date and time values is established in two ways:

- Each entity that participates in the relationship contains the same `datetime_link` attribute.
- The literal string that spans the complete date and time mention, when they are mentioned together in input, is the same for both entities. The location information is appended to the `datetime_link` name.

Recognized formats

- now
- two hours from now

- on Monday at 4pm

Attributes

- `.datetime_link`: If present, this indicates that the user's input mentions a date and time together, which implies that the date and time are related to one another. The start and end index values of the string that includes the date and time are saved as part of the link name.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
`@sys-date.datetime_link` and `@sys-time.datetime_link`	string	If the input is, `Are you open today at 5?`, then `today` is recognized as a date mention (`2020-05-21`) at location `[13,18]` and `at 5` is recognized as a time mention (`05:00:00`) at location `[19,23]`. A location value of `[13,23]`, which spans both the date and time mentions, is appended to the resulting `datetime_link`. It is named `datetime_link_13_23` and it is included in the output of the `@sys-date` and `@sys-time` entities that participate in the relationship.

Date and time attributes

Time zones

Mentions of a date or time that are relative to the current time are resolved for a chosen time zone. By default, the time zone is Greenwich mean time. Therefore, REST API clients that are located in different time zones get the current Coordinated Universal Time when `now` is mentioned in input.

Optionally, the REST API client can add the local time zone as the context variable `$timezone`. This context variable must be sent with every client request. For example, the `$timezone` value can be `America/Los_Angeles`, `EST`, or `UTC`. For a full list of supported time zones, see Supported time zones.

When the `$timezone` variable is provided, the values of relative `@sys-date` and `@sys-time` mentions are computed based on the client time zone instead of Coordinated Universal Time.

Tip: For information about processing date and time values, see the Date and time method reference.

@sys-number entity

The `@sys-number` system entity detects mentions of numbers in user input. The number can be written with either numerals or words. In either case, a number is returned.

Recognized formats

- 21
- twenty one
- 3.13

Attributes

- `.literal`: Exact phrase in input that is interpreted to be the number.
- `.location`: Index element values of the first and last letters in the text string that is interpreted to be a number.
- `.numeric_value`: Canonical numeric value as an integer or a double.
- `.range_link`: If present, indicates that the user's input contains a number range. The location value of the text that specifies the range is appended to the link name. Additional information is provided, including the role that each `@sys-number` plays in the range relationship. For example, the start number has a role type of `number_from` and the end number has a role type of `number_to`. To check a role value, you can use the syntax `@sys-number.role?.type == 'number_from'`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-number	string	If the input is `twenty`, returns `20`. If the input is `1,234.56`, returns `1234.56`.
@sys-number.literal	string	If the input is `twenty`, returns `twenty`. If the input is `1,234.56` returns `1,234.56`.
@sys-number.location	array	If the input is `twenty`, returns [0,6]. If the input is `1,234.56`, returns [0,8]

@sys-number.numeric_value	number	If the input is `twenty`, returns `20`. If the input is `1,234.56`, returns `1234.56`.
@sys-number.range_link	string	If the input is `I'm interested in 5 to 7.`. Each of the two `@sys-number` system entities that are detected have a `range_link` of `number_range_18_24` in their output. The first `@sys-number` entity (`5`) has a `number_from` role type. The last `@sys-number` entity (`7`) has a `number_to` role type.

@sys-number examples

@sys-number usage tips

- If you use the @sys-number entity as a node condition and the user specifies zero as the value, the 0 value is recognized properly as a number. However, the 0 is interpreted as a `null` value for the condition, which results in the node not being processed. To check for numbers in a way that handles zeros properly, use the expression `@sys-number == 0` in the node condition also. The full expression to use is `@sys-number OR @sys-number == 0`.

- If you use @sys-number to compare number values in a condition, be sure to separately include a check for the presence of a number itself. If no number is found, @sys-number evaluates to null. Your comparison might evaluate to true even when no number is present.

  For example, do not use `@sys-number<4` alone because if no number is found, `@sys-number` evaluates to null. Because null is less than 4, the condition evaluates to true even though no number is present.

  Use `(@sys-number OR @sys-number == 0) AND @sys-number < 4` instead. If no number is present, the first condition evaluates to false. As a result, the whole condition evaluates to false.

Tip: For more information about processing number values, see the Numbers method reference.

@sys-percentage entity

The `@sys-percentage` system entity detects mentions of percentages in user input. The percentage can be expressed in an utterance with the percent symbol or written out using the word `percent`. In either case, a numeric value is returned.

Recognized formats

- 15%
- 10 percent

Attributes

- `.literal`: Exact phrase in input that is interpreted to be a percentage.
- `.location`: Index element values of the first and last letters in the text string that is interpreted to be a percentage mention.
- `.numeric_value`: Canonical numeric value as an integer or a double
- `.range_link`: If present, indicates that the user's input contains a range, such as `from 2 to 3 percent`. The location value of the text that specifies the range is appended to the link name. Additional information is provided, including the role that each `@sys-percentage` plays in the range relationship. For example, the start number has a role type of `number_from` and the end number has a role type of `number_to`. To check a role value, you can use the syntax `@sys-percentage.role?.type == 'number_from'`.

The following table illustrates the information that each attribute captures from the user input.

Attribute	Type	Example
@sys-percentage	string	If input contains `1,234.56%`, returns `1234.56`.
@sys-percentage.literal	string	If input contains `50 percent` returns `50 percent`.
@sys-percentage.location	array	If input contains `from 2 to 3 percent`, returns `[0,19]`.
@sys-percentage.numeric_value	number	If input contains `50 percent` returns `50`.
@sys-percentage.range_link	string	If input contains `from 2 to 3 percent`, returns `number_range_0_19`.

@sys-percentage examples

@sys-percentage usage tips

- If you use the @sys-percentage entity as a node condition and the user specifies `0%` as the value, the value is recognized as a percentage properly, but the condition is evaluated to the number zero not the percentage 0%. As a result, the `null` in the condition is evaluated to false and the node is not processed. To check for percentages in a way that handles zero percentages properly, use the expression `@sys-percentage OR @sys-percentage == 0` in the node condition instead.
- If you input a value like `1-2%`, the values `1%` and `2%` are returned as system entities.

Using system entities in conditions

In a node that conditions on the `#Customer_Care_Store_Hours` intent, you can add conditional responses that use new system entity properties to provide slightly different answers about store hours depending on what the user asks.

Tip: You probably do not want to use all of these conditional responses in a real dialog; they are described here merely to illustrate what's possible.

Conditional response condition syntax	Description	Example response text
`@sys-date.festival == 'thanksgiving'`	Checks whether the customer is asking about hours on Thanksgiving day in particular.	We give out free meals to those in need on Thanksgiving Day.
`@sys-date.festival`	Checks whether the customer is asking about hours on another specific holiday.	We are closed on Christmas Day, July 4th, and President's Day. On other holidays, we are open from 10AM to 5PM.
`@sys-time.part_of_day == 'night'`	Checks whether the user includes any terms that mention night as the time of day in the input.	We are not open late; we close at 9PM most days.
`@sys-date.datetime_link && input.text.contains('today') && now().sameOrAfter(@sys-time)`	Checks whether the input contains a phrase such as `today at 8`. It also checks whether the `@sys-time` that is detected is earlier than the current time of day. You can add a context variable to the conditional response that creates a `$new_time` variable with the value `<? @sys-time.plusHours(12) ?>`. You can use this approach to create a date variable that captures the more likely time that is meant by a user who, at noon time asks, `Are you open today at 8`. The system assumes that the user means 8AM instead of 8PM.	You mean today at $new_time, correct?
`@sys-time.range_link && entities[0].role?.type == 'time_from' && entities[1].role?.type == 'time_to'`	Checks whether the input contains a time range. The condition also makes sure that the first `@sys-time` system entity that is listed in the entities array is the start time and that the second one is the end time before including them in the response.	You want to know if we're open between `<? entities[0] ?>` and `<? entities[1] ?>`, is that correct?
`@sys-time && entities.size() < 2 && entities[0].role?.type == 'time_to'`	Checks whether the input contains one `@sys-time` mention that has a `time_to` role type. If the user input matches this condition, it suggests that the user specified the end time of an open-ended time range. For example, the response would be triggered by the input `Are you open until 9pm?`. This input matches because it identifies only one time in a time span, and the mention has a `time_to` role.	Do you mean from now (`<? now().reformatDateTime('h:mm a') ?>`) until `<? @sys-time.reformatDateTime('h:mm a') ?>`?
`@sys-date.day_of_week == 'sunday'`	Checks whether a specific date that the user is asking about falls on a Sunday.	We are closed on Sundays.
`@sys-date.specific_day_of_week == 'monday' && @sys-date.alternative`	Checks whether the user mentioned the weekday `Monday` in their query. For example, `Are you open Monday`. The condition also checks whether any alternative dates were stored. Alternative dates are created when your assistant isn't entirely sure which Monday the user means, so it stores the dates of alternative Mondays also. When a user specifies a weekday, your assistant assumes that the user means the future occurrence of the day (the coming Monday). You can add a response that double checks the intended date, by using the detected alternative value. In this case, the alternative date is the previous Monday's date.	Do you mean `@sys-date` or `@sys-date.alternative`?

true	Responds to any other requests for store hour information.	We're open from 9AM to 9PM Monday through Saturday.

System entities in conditional responses

Time zones supported by system entities

The following list of supported time zones can be used with the time zone functions for the @sys-date and @sys-time entities.

Time zone	Time zone
Africa/Abidjan	Africa/Accra
Africa/Addis_Ababa	Africa/Algiers
Africa/Asmara	Africa/Asmera
Africa/Bamako	Africa/Bangui
Africa/Banjul	Africa/Bissau
Africa/Blantyre	Africa/Brazzaville
Africa/Bujumbura	Africa/Cairo
Africa/Casablanca	Africa/Ceuta
Africa/Conakry	Africa/Dakar
Africa/Dar_es_Salaam	Africa/Djibouti
Africa/Douala	Africa/El_Aaiun
Africa/Freetown	Africa/Gaborone
Africa/Harare	Africa/Johannesburg
Africa/Juba	Africa/Kampala
Africa/Khartoum	Africa/Kigali
Africa/Kinshasa	Africa/Lagos
Africa/Libreville	Africa/Lome
Africa/Luanda	Africa/Lubumbashi
Africa/Lusaka	Africa/Malabo
Africa/Maputo	Africa/Maseru
Africa/Mbabane	Africa/Mogadishu
Africa/Monrovia	Africa/Nairobi
Africa/Ndjamena	Africa/Niamey

Africa/Nouakchott	Africa/Ouagadougou
Africa/Porto-Novo	Africa/Sao_Tome
Africa/Timbuktu	Africa/Tripoli
Africa/Tunis	Africa/Windhoek
America/Adak	America/Anchorage
America/Anguilla	America/Antigua
America/Araguaina	America/Argentina/Buenos_Aires
America/Argentina/Catamarca	America/Argentina/ComodRivadavia
America/Argentina/Cordoba	America/Argentina/Jujuy
America/Argentina/La_Rioja	America/Argentina/Mendoza
America/Argentina/Rio_Gallegos	America/Argentina/Salta
America/Argentina/San_Juan	America/Argentina/San_Luis
America/Argentina/Tucuman	America/Argentina/Ushuaia
America/Aruba	America/Asuncion
America/Atikokan	America/Atka
America/Bahia	America/Bahia_Banderas
America/Barbados	America/Belem
America/Belize	America/Blanc-Sablon
America/Boa_Vista	America/Bogota
America/Boise	America/Buenos_Aires
America/Cambridge_Bay	America/Campo_Grande
America/Cancun	America/Caracas
America/Catamarca	America/Cayenne
America/Cayman	America/Chicago
America/Chihuahua	America/Coral_Harbour
America/Cordoba	America/Costa_Rica
America/Creston	America/Cuiaba
America/Curacao	America/Danmarkshavn

America/Dawson	America/Dawson_Creek
America/Denver	America/Detroit
America/Dominica	America/Edmonton
America/Eirunepe	America/El_Salvador
America/Ensenada	America/Fort_Nelson
America/Fort_Wayne	America/Fortaleza
America/Glace_Bay	America/Godthab
America/Goose_Bay	America/Grand_Turk
America/Grenada	America/Guadeloupe
America/Guatemala	America/Guayaquil
America/Guyana	America/Halifax
America/Havana	America/Hermosillo
America/Indiana/Indianapolis	America/Indiana/Knox
America/Indiana/Marengo	America/Indiana/Petersburg
America/Indiana/Tell_City	America/Indiana/Vevay
America/Indiana/Vincennes	America/Indiana/Winamac
America/Indianapolis	America/Inuvik
America/Iqaluit	America/Jamaica
America/Jujuy	America/Juneau
America/Kentucky/Louisville	America/Kentucky/Monticello
America/Knox_IN	America/Kralendijk
America/La_Paz	America/Lima
America/Los_Angeles	America/Louisville
America/Lower_Princes	America/Maceio
America/Managua	America/Manaus
America/Marigot	America/Martinique
America/Matamoros	America/Mazatlan
America/Mendoza	America/Menominee

America/Merida	America/Metlakatla
America/Mexico_City	America/Miquelon
America/Moncton	America/Monterrey
America/Montevideo	America/Montreal
America/Montserrat	America/Nassau
America/New_York	America/Nipigon
America/Nome	America/Noronha
America/North_Dakota/Beulah	America/North_Dakota/Center
America/North_Dakota/New_Salem	America/Ojinaga
America/Panama	America/Pangnirtung
America/Paramaribo	America/Phoenix
America/Port-au-Prince	America/Port_of_Spain
America/Porto_Acre	America/Porto_Velho
America/Puerto_Rico	America/Rainy_River
America/Rankin_Inlet	America/Recife
America/Regina	America/Resolute
America/Rio_Branco	America/Rosario
America/Santa_Isabel	America/Santarem
America/Santiago	America/Santo_Domingo
America/Sao_Paulo	America/Scoresbysund
America/Shiprock	America/Sitka
America/St_Barthelemy	America/St_Johns
America/St_Kitts	America/St_Lucia
America/St_Thomas	America/St_Vincent
America/Swift_Current	America/Tegucigalpa
America/Thule	America/Thunder_Bay
America/Tijuana	America/Toronto
America/Tortola	America/Vancouver

America/Virgin	America/Whitehorse
America/Winnipeg	America/Yakutat
America/Yellowknife	Antarctica/Casey
Antarctica/Davis	Antarctica/DumontDUrville
Antarctica/Macquarie	Antarctica/Mawson
Antarctica/McMurdo	Antarctica/Palmer
Antarctica/Rothera	Antarctica/South_Pole
Antarctica/Syowa	Antarctica/Troll
Antarctica/Vostok	Arctic/Longyearbyen
Asia/Aden	Asia/Almaty
Asia/Amman	Asia/Anadyr
Asia/Aqtau	Asia/Aqtobe
Asia/Ashgabat	Asia/Ashkhabad
Asia/Baghdad	Asia/Bahrain
Asia/Baku	Asia/Bangkok
Asia/Beirut	Asia/Bishkek
Asia/Brunei	Asia/Calcutta
Asia/Chita	Asia/Choibalsan
Asia/Chongqing	Asia/Chungking
Asia/Colombo	Asia/Dacca
Asia/Damascus	Asia/Dhaka
Asia/Dili	Asia/Dubai
Asia/Dushanbe	Asia/Gaza
Asia/Harbin	Asia/Hebron
Asia/Ho_Chi_Minh	Asia/Hong_Kong
Asia/Hovd	Asia/Irkutsk
Asia/Istanbul	Asia/Jakarta
Asia/Jayapura	Asia/Jerusalem

Asia/Kabul	Asia/Kamchatka
Asia/Karachi	Asia/Kashgar
Asia/Kathmandu	Asia/Katmandu
Asia/Khandyga	Asia/Kolkata
Asia/Krasnoyarsk	Asia/Kuala_Lumpur
Asia/Kuching	Asia/Kuwait
Asia/Macao	Asia/Macau
Asia/Magadan	Asia/Makassar
Asia/Manila	Asia/Muscat
Asia/Nicosia	Asia/Novokuznetsk
Asia/Novosibirsk	Asia/Omsk
Asia/Oral	Asia/Phnom_Penh
Asia/Pontianak	Asia/Pyongyang
Asia/Qatar	Asia/Qyzylorda
Asia/Rangoon	Asia/Riyadh
Asia/Saigon	Asia/Sakhalin
Asia/Samarkand	Asia/Seoul
Asia/Shanghai	Asia/Singapore
Asia/Srednekolymsk	Asia/Taipei
Asia/Tashkent	Asia/Tbilisi
Asia/Tehran	Asia/Tel_Aviv
Asia/Thimbu	Asia/Thimphu
Asia/Tokyo	Asia/Ujung_Pandang
Asia/Ulaanbaatar	Asia/Ulan_Bator
Asia/Urumqi	Asia/Ust-Nera
Asia/Vientiane	Asia/Vladivostok
Asia/Yakutsk	Asia/Yekaterinburg
Asia/Yerevan	Atlantic/Azores

Atlantic/Bermuda	Atlantic/Canary
Atlantic/Cape_Verde	Atlantic/Faeroe
Atlantic/Faroe	Atlantic/Jan_Mayen
Atlantic/Madeira	Atlantic/Reykjavik
Atlantic/South_Georgia	Atlantic/St_Helena
Atlantic/Stanley	Australia/ACT
Australia/Adelaide	Australia/Brisbane
Australia/Broken_Hill	Australia/Canberra
Australia/Currie	Australia/Darwin
Australia/Eucla	Australia/Hobart
Australia/LHI	Australia/Lindeman
Australia/Lord_Howe	Australia/Melbourne
Australia/NSW	Australia/North
Australia/Perth	Australia/Queensland
Australia/South	Australia/Sydney
Australia/Tasmania	Australia/Victoria
Australia/West	Australia/Yancowinna
Brazil/Acre	Brazil/DeNoronha
Brazil/East	Brazil/West
CET	CST6CDT
Canada/Atlantic	Canada/Central
Canada/East-Saskatchewan	Canada/Eastern
Canada/Mountain	Canada/Newfoundland
Canada/Pacific	Canada/Saskatchewan
Canada/Yukon	Chile/Continental
Chile/EasterIsland	Cuba
EET	EST5EDT
Egypt	Eire

Etc/GMT	Etc/GMT+0
Etc/GMT+1	Etc/GMT+10
Etc/GMT+11	Etc/GMT+12
Etc/GMT+2	Etc/GMT+3
Etc/GMT+4	Etc/GMT+5
Etc/GMT+6	Etc/GMT+7
Etc/GMT+8	Etc/GMT+9
Etc/GMT-0	Etc/GMT-1
Etc/GMT-10	Etc/GMT-11
Etc/GMT-12	Etc/GMT-13
Etc/GMT-14	Etc/GMT-2
Etc/GMT-3	Etc/GMT-4
Etc/GMT-5	Etc/GMT-6
Etc/GMT-7	Etc/GMT-8
Etc/GMT-9	Etc/GMT0
Etc/Greenwich	Etc/UCT
Etc/UTC	Etc/Universal
Etc/Zulu	Europe/Amsterdam
Europe/Andorra	Europe/Athens
Europe/Belfast	Europe/Belgrade
Europe/Berlin	Europe/Bratislava
Europe/Brussels	Europe/Bucharest
Europe/Budapest	Europe/Busingen
Europe/Chisinau	Europe/Copenhagen
Europe/Dublin	Europe/Gibraltar
Europe/Guernsey	Europe/Helsinki
Europe/Isle_of_Man	Europe/Istanbul
Europe/Jersey	Europe/Kaliningrad

Europe/Kiev	Europe/Lisbon
Europe/Ljubljana	Europe/London
Europe/Luxembourg	Europe/Madrid
Europe/Malta	Europe/Mariehamn
Europe/Minsk	Europe/Monaco
Europe/Moscow	Europe/Nicosia
Europe/Oslo	Europe/Paris
Europe/Podgorica	Europe/Prague
Europe/Riga	Europe/Rome
Europe/Samara	Europe/San_Marino
Europe/Sarajevo	Europe/Simferopol
Europe/Skopje	Europe/Sofia
Europe/Stockholm	Europe/Tallinn
Europe/Tirane	Europe/Tiraspol
Europe/Uzhgorod	Europe/Vaduz
Europe/Vatican	Europe/Vienna
Europe/Vilnius	Europe/Volgograd
Europe/Warsaw	Europe/Zagreb
Europe/Zaporozhye	Europe/Zurich
GB	GB-Eire
GMT	GMT0
Greenwich	Hongkong
Iceland	Indian/Antananarivo
Indian/Chagos	Indian/Christmas
Indian/Cocos	Indian/Comoro
Indian/Kerguelen	Indian/Mahe
Indian/Maldives	Indian/Mauritius
Indian/Mayotte	Indian/Reunion

Iran	Israel
Jamaica	Japan
Kwajalein	Libya
MET	MST7MDT
Mexico/BajaNorte	Mexico/BajaSur
Mexico/General	NZ
NZ-CHAT	Navajo
PRC	PST8PDT
Pacific/Apia	Pacific/Auckland
Pacific/Bougainville	Pacific/Chatham
Pacific/Chuuk	Pacific/Easter
Pacific/Efate	Pacific/Enderbury
Pacific/Fakaofo	Pacific/Fiji
Pacific/Funafuti	Pacific/Galapagos
Pacific/Gambier	Pacific/Guadalcanal
Pacific/Guam	Pacific/Honolulu
Pacific/Johnston	Pacific/Kiritimati
Pacific/Kosrae	Pacific/Kwajalein
Pacific/Majuro	Pacific/Marquesas
Pacific/Midway	Pacific/Nauru
Pacific/Niue	Pacific/Norfolk
Pacific/Noumea	Pacific/Pago_Pago
Pacific/Palau	Pacific/Pitcairn
Pacific/Pohnpei	Pacific/Ponape
Pacific/Port_Moresby	Pacific/Rarotonga
Pacific/Saipan	Pacific/Samoa
Pacific/Tahiti	Pacific/Tarawa
Pacific/Tongatapu	Pacific/Truk

Pacific/Wake	Pacific/Wallis
Pacific/Yap	Poland
Portugal	ROK
Singapore	SystemV/AST4
SystemV/AST4ADT	SystemV/CST6
SystemV/CST6CDT	SystemV/EST5
SystemV/EST5EDT	SystemV/HST10
SystemV/MST7	SystemV/MST7MDT
SystemV/PST8	SystemV/PST8PDT
SystemV/YST9	SystemV/YST9YDT
Turkey	UCT
US/Alaska	US/Aleutian
US/Arizona	US/Central
US/East-Indiana	US/Eastern
US/Hawaii	US/Indiana-Starke
US/Michigan	US/Mountain
US/Pacific	US/Pacific-New
US/Samoa	UTC
Universal	W-SU
WET	Zulu
EST	HST
MST	ACT
AET	AGT
ART	AST
BET	BST
CAT	CNT
CST	CTT
EAT	ECT

IET	IST
JST	MIT
NET	NST
PLT	PNT
PRT	PST
SST	VST

Supported time zones

Festivals

The `@sys-date` system entity recognizes dates that fall on national holidays in different geographic regions.

The following sections list the holidays that `@sys-date` recognizes for different locales.

Holidays (en-us)

Holiday name	String to use to check for a festival match
New Year's Day	`newyear`
Inauguration Day	`inauguration`
Martin Luther King, Jr Day	`luther`
Groundhog Day	`groundhog`
Washington's Birthday	`washington`
President's Day	`president`
Valentine's Day	`valentine`
International Women's Day	`women`
St Patrick's Day	`patrick`
April Fool's Day	`fool`
Good Friday	`goodfriday`
Easter Sunday	`easter`
Easter Monday	`eastermonday`
Earth Day	`earth`
Memorial Day	`memorial`
Mother's Day	`mother`
Father's Day	`father`

Independence Day	independence
Labor Day	labor
Columbus Day	columbus
Veteran's Day	veterans
Thanksgiving	thanksgiving
Halloween	halloween
Christmas Eve	christmaseve
Christmas	christmas
Boxing Day	boxing
New Year's Eve	newyearseve

US English holidays

Holidays (en-ca)

Holiday name	String to use to check for a festival match
New Year's Day	newyear
Epiphany	epiphany
Groundhog Day	groundhog
Valentines Day	valentine
International Women's Day	women
Commonwealth Day	commonwealth
St Patrick's Day	patrick
April Fool's Day	fool
Good Friday	goodfriday
Easter Sunday	easter
Easter Monday	eastermonday
National Tartan Day	tartan
Vimy Ridge Day	vimyridge
Earth Day	earth
Mother's Day	mother
Victoria Day	victoria

Father's Day	father
National Indigenous Day	indigenous
Canada Day	canada
Labor Day	labor
Thanksgiving	thanksgiving
Remembrance Day	remembrance
Halloween	halloween
Anniversary of the Statue of Westminster	westminster
Christmas Eve	christmaseve
Christmas Day	christmas
Boxing Day	boxing
New Year's Eve	newyearseve

Canadian holidays

Holidays (pt-br)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Carnival Friday	carnivalday
Carnival Friday	carnivalfriday
Carnival Saturday	carnivalsaturday
Carnival Sunday	carnivalsunday
Carnival Monday	carnivalmonday
Carnival Tuesday	carnivaltuesday
Carnival End (Until 2pm)	carnivalend
Good Friday	goodfriday
Tiradentes Day	tiradentesday
Easter Sunday	eastersunday
Labor Day / May Day	laborday
Mother's Day	mothersday
Brazilian Valentine's Day	valentinesday

Corpus Christi	corpuschristi
Father's Day	fatherday
Independence Day	independenceday
Our Lady of Aparecida / Children's Day	ourladyofaparecida
Teacher's Day	teachersday
Public Service Holiday	publicserviceholiday
All Souls' Day	allsoulsday
Republic Proclamation Day	republicproclamationday
Black Consciousness Day	blackconsciousnessday
Christmas Eve (from 2pm)	christmaseve
Christmas Day	christmasday
New Year's Eve (from 2pm)	newyearseve

Brazilian holidays

Holidays (en-gb)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
2nd January (Scotland)	2ndjanuary
St. David's Day (Wales)	st.davidsday
Valentines Day	valentine
St Patrick's Day (Northern Ireland)	stpatricksday
St Patrick's Day Off (Northern Ireland)	stpatricksdayoff
Good Friday	goodfriday
Easter Sunday	easter
Easter Monday (England, Wales, Northern Ireland, Scotland)	eastermonday
St. George's Day	st.georgesday
Early May Bank Holiday	earlymaybankholiday
Spring Bank Holiday	springbankholiday
Queen's Birthday	queensbirthday
Battle of the Boyne (Northern Ireland)	battleoftheboyne

Summer Bank Holiday (Scotland)	summerbankholiday
Summer Bank Holiday (England, Wales, Northern Ireland, Scotland)	summerbankholiday
Halloween	halloween
Guy Fawkes Day	guyfawkesday
Rememberance Sunday	rememberancesunday
St Andrew's Day (Scotland)	standrewsday
St Andrew's Day Observed (Scotland)	standrewsdayobserved
Christmas Day	christmas
Boxing Day	boxingday
Battle of the Boyne Observed (Northern Ireland)	battleoftheboyneobserved
Bank Holiday (Day 1)	bankholidayfirst
Bank Holiday (Day 2)	bankholidaysecond
2nd January (substitute day) (Scotland)	2ndjanuary
New Year's Eve	newyearseve
'New Year's Day' Observed	newyearsdayobserved

British English holidays

Holidays (cs-cz)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Restoration of the Czech Independence Day	restorationofczechindependenceday
St. Valentine's Day	valentinesday
International Women's Day	internationalwomensday
Good Friday	goodfriday
Easter Monday	eastermonday
April Fools Day	aprilfools
Labor Day / May Day	laborday
Victory in Europe Day	victoryineuropeday
Mother's Day	mothersday
Children's Day	childrensday

Father's Day	fathersday
Saints Cyril and Methodius	saintcyrilandmethodius
Jan Hus Day	janhusday
St. Wenceslas Day	wenceslasday
Independent Czechoslovak State Day	independentczechoslovakstateday
Struggle for Freedom and Democracy Day	struggleforfreedomanddemocracyday
Christmas Eve	christmaseve
Christmas Day	christmasday
St. Stephen's Day	st.stephensday
New Year's Eve	newyearseve

Czech holidays

Holidays (nl-nl)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Valentine's Day	Valentinesday
Good Friday	goodfriday
Easter Sunday	eastersunday
Easter Monday	eastermonday
King's Birthday	kingsbirthday
Liberation Day	liberationday
Ascension Day	ascensionday
Whit Sunday	whitsunday
Whit Monday	whitmonday
St Nicholas' Eve/Sinterklaas	stnicholaseve
Christmas Eve	christmaseve
Christmas Day	christmasday
Second Day of Christmas	seconddayofchristmas
New Year's Eve	newyearseve

Dutch holidays

Holidays (fr-fr)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Valentine's Day	valentinesday
Easter Sunday	eastersunday
Easter Monday	eastermonday
Labor Day / May Day	laborday
WWII Victory Day	wwiivictoryday
Mother's Day	mothersday
Ascension Day	ascensionday
Whit Sunday	whitsunday
Whit Monday	whitmonday
Father's Day	fathersday
Bastille Day	bastilleday
Assumption of Mary	assumptionofmary
All Saints' Day	allsaintsday
Armistice Day	armisticeday
Christmas Eve	christmaseve
Christmas Day	christmasday
New Year's Eve	newyearseve

French holidays

Holidays (de-de)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Epiphany (BW, BY, ST)	epiphany
Valentine's Day	valentinesday
Shrove Monday	shrovemonday
Carnival / Shrove Tuesday	carnival
Carnival / Ash Wednesday	carnival

International Women's Day (Most regions)	internationalwomensday
April Fool's Day	aprilfools
Palm Sunday	palmsunday
Maundy Thursday	maundythursday
Good Friday	goodfriday
Easter Sunday (Most regions)	eastersunday
Easter Monday	eastermonday
May Day	mayday
Mother's Day	mothersday
Father's Day	fathersday
Ascension Day	ascensionday
Whit Sunday (Most regions)	whitsunday
Whit Sunday (Brandenburg)	whitsunday
Whit Monday	whitmonday
Corpus Christi (Many regions)	corpuschristi
Assumption of Mary (Bavaria, Saarland)	assumptioofmary
Day of German Unity	dayofgermanunity
Reformation Day (Most regions)	reformationday
Halloween	halloween
All Saints' Day (Many regions)	allsaintsday
St Martin's Day	st.martinsday
National Day of Mourning	nationaldayofmourning
Repentance Day (Saxony)	repentanceday
Sunday of the Dead	sundayofthedead
First Advent Sunday	firstadventsunday
Saint Nicholas Day	saintnicholasday
Second Advent Sunday	secondadventsunday
Third Advent Sunday	thirdadventsunday

Fourth Advent Sunday	fourthadventsunday
Christmas Day	christmasday
Boxing Day	boxingday
Whit Sunday (Most regions)	whitsunday
New Year's Eve	newyearseve

German holidays

Holidays (it-it)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Valentine's Day	valentineday
Epiphany	epiphany
Good Friday	goodfriday
Easter Sunday	eastersunday
Easter Monday	eastermonday
Liberation Day	liberationday
Labor Day / May Day	laborday
Republic Day	republicday
Assumption of Mary / Ferragosto	assumptionofmary
All Saints' Day	allsaintsday
Feast of the Immaculate Conception	feastoftheimmaculateconception
Christmas Day	christmasday
St Stephen's Day	st.stephensday
Labor Day / May Day	laborday
New Year's Eve	newyearseve

Italian holidays

Holidays (pt-pt)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Valentine's Day	valentinesday

Carnival / Shrove Tuesday	carnivalday
Good Friday	goodfriday
Easter Sunday	eastersunday
Liberty Day	libertyday
Labor Day / May Day	laborday
Mother's Day	mothersday
Father's Day	fathersday
Portugal Day	portugalday
Corpus Christi	corpuschristi
Assumption of Mary	assumptionofmary
Republic Implantation	republicimplantation
All Saints' Day	allsaintsday
Restoration of Independence	restorationofindependence
Feast of the Immaculate Conception	immaculateconception
Christmas Eve	christmaseve
Christmas Day	christmasday
New Year's Eve	newyearseve

Portuguese holidays

Holidays (sk-sk)

Holiday name	String to use to check for a festival match
Republic Day	republicday
Epiphany	epiphany
Good Friday	goodfriday
Easter Sunday	eastersunday
Easter Monday	eastermonday
Labor Day	laborday
End of World War II	endofworldwarii
St. Cyril & St. Methodius Day	st.cyril&st.methodiusday
National Uprising Day	nationaluprisingday

Constitution Day	constitutionday
Day of Our Lady of Sorrows	dayofourladyofsorrows
All Saints' Day	allsaintsday
Fight for Freedom and Democracy Day	fightforfreedomanddemocracyday
Christmas Eve	christmaseve
Christmas Day	christmasday
St. Stephen's Day	st.stephensday
St. Cyril & St. Methodius Day	cyrilmethodiusday
New Year's Eve	newyearseve

Slovak holidays

Holidays (es-es)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Epiphany	epiphany
Epiphany observed (Many regions)	epiphanyobserved
Valentine's Day	valentinesday
Andalusia Day	andalusiaday
Baleares Day	balearesday
Ash Wednesday	ashwednesday
San Jose (Many regions)	sanjose
Palm Sunday	palmsunday
Maundy Thursday (Many regions)	maundythursday
Good Friday	goodfriday
Easter Sunday	eastersunday
Easter Monday (Many regions)	eastermonday
Aragon Day	aragonday
Saint George Day	stgeorgeday
Labor Day / May Day	laborday
Community Day	communityday

San Isidro	saintisidro
Mothers' Day	mothersday
Whit Sunday/Pentecost	whitsunday
Feast of Saint James the Apostle (GA, PV, S)	feastofsaintjamestheapostle
Assumption of Mary	assumptionofmary
Hispanic Day	hispanicday
All Saints' Day (All)	allsaintsday
Constitution Day	constitutionday
Immaculate Conception	immaculateconception
Immaculate Conception observed (Many regions)	immaculateconceptionobserved
Christmas Eve	christmaseve
Christmas Day	christmasday
Feast of the Holy Family	feastoftheholyfamily
Corpus Christi	corpuschristi
All Saints' Day' observed (Most regions)	allsaintsdayobserved
Assumption observed	assumptionobserved
Cantabria Day	cantabriaday
Christmas Day observed (Balearic Islands)	christmasdayobserved
Labor Day observed (Many regions)	labordayobserved
Christmas Day observed (Many regions)	christmasdayobserved
Immaculate Conception observed	immaculateconceptionobserved
Hispanic Day observed (Many regions)	hispanicdayobserved
Almuneda Day	almunedaday
New Year's Eve	newyearseve

Spanish holidays

Holidays (ar-ar)

Holiday name	String to use to check for a festival match
New Year	newyear
Valentine	valentine

Easter Sunday	`easter`
Christmas Eve	`christmaseve`
Christmas	`christmas`
Eid al-Fitr	`eidalfitr`
Eid al-Adha	`eidaladha`
Ramadan	`ramadan`
Islamic New Year	`islamicnewyear`
Ashura	`ashura`
Mawlid an-Nabi	`mawlidannabi`
Day of Arafat	`dayofarafat`
Laylat al-Miraj	`laylatalmiraj`

Arabic holidays

Holidays (iw-il)

Holiday name	String to use to check for a festival match
Erev Purim	`erevpurim`
Yom HaAliyah	`yomhaaliyah`
Erev Pesach	`erevpesach`
Pesach I (First day of Passover)	`pesachi`
Pesach II (Passover)	`pesachii`
Pesach III (Passover)	`pesachiii`
Pesach IV (Passover)	`pesachiv`
Pesach V (Passover)	`pesachv`
Pesach VI (Passover)	`pesachvi`
Pesach VII (Last day of Passover)	`pesachvii`
Yom HaShoah/Holocaust Memorial Day	`yomhashoah`
Yom HaZikaron (Memorial Day)	`yomhazikaron`
Yom HaAtzmaut (Independence Day)	`yomhaatzmaut`
Yom Yerushalayim (Jerusalem Day)	`yomyerushalayim`
Erev Shavuot	`erevshavuot`

Shavuot (Pentecost)	shavuot
Erev Tisha B'Av	erevtishabav
Tisha B'Av	tishabav
Erev Rosh Hashana	erevroshhashana
Rosh Hashana (New Year)	roshhashana
Rosh Hashana II (New Year day 2)	roshhashanaii
Erev Yom Kippur	erevyomkippur
Yom Kippur	yomkippur
Erev Sukkot	erevsukkot
Sukkot I	sukkoti
Sukkot II	sukkotii
Sukkot III	sukkotiii
Sukkot IV	sukkotiv
Sukkot V	sukkotv
Sukkot VI	sukkotvi
Sukkot VII/Hoshanah Rabah	sukkotvii
Shmini Atzeret/Simchat Torah	shminiatzeret
Yom HaAliyah School Observance	yomhaaliyahschoolobservance
Hanukkah I (Holiday of lights)	hanukkahi
Hanukkah II	hanukkahii
Hanukkah III	hanukkahiii
Hanukkah IV	hanukkahiv
Hanukkah V	hanukkahv
Hanukkah VI/Rosh Chodesh Tevet	hanukkahvi
Hanukkah VII	hanukkahvii
Hanukkah VIII	hanukkahviii

Hebrew holidays

Holidays (zh-cn)

Holiday name	String to use to check for a festival match

New Year's Day	newyearsday
New Year's Weekend	newyearsweekend
Double Seventh Festival	doubleseventhfestival
Double Ninth Festival	doubleninthfestival
Laba Festival	labafestival
Ching Ming Festival	chingmingfestival
Longtaitou Festival	longtaitoufestival
Spring Festival Eve	springfestivaleve
Chinese New Year Eve	chinesenewyeareve
Chinese New Year	chinesenewyear
Lantern Festival	lanternfestival
Zhonghe Festival	zhonghefestival
Christmas	christmas
Labor Day	laborday
Valentine	valentine
Easter Sunday	easter
Dragon Boat Festival	dragonboatfestival
Mid-Autumn Festival	midautumnfestival
National Day	nationalday

Simplified Chinese holidays

Holidays (zh-tw)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
New Year's weekend	newyearsweekend
Double Seventh Festival	doubleseventhfestival
Double Ninth Festival	doubleninthfestival
Laba Festival	labafestival
Ching Ming Festival	chingmingfestival
Longtaitou Festival	longtaitoufestival

Spring Festival Eve	springfestivaleve
Chinese New Year Eve	chinesenewyeareve
Chinese New Year	chinesenewyear
Lantern Festival	lanternfestival
Zhonghe Festival	zhonghefestival
Christmas	christmas
Labor Day	laborday
Valentine	valentine
easter sunday	easter
Dragon Boat Festival	dragonboatfestival
Mid-Autumn Festival	midautumnfestival
National Day	nationalday

Traditional Chinese holidays

Holidays (ja-jp)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
January 2 Bank Holiday	january2bankholiday
January 3 Bank Holiday	january3bankholiday
Coming of Age Day	comingofageday
National Foundation Day	nationalfoundationday
Easter Sunday	easter
Saint Valentines Day	valentinesday
Dolls' Festival/Girls' Festival	dollsfestival
Spring Equinox	springequinox
Shōwa Day	showaday
Coronation Day holiday	coronationdayholiday
Coronation Day	coronationday
Constitution Memorial Day	constitutionmemorialday
Greenery Day	greeneryday

Children's Day	childrensday
'Children's Day' observed	childrensdayobserved
Star Festival	starfestival
Sea Day	seaday
Mountain Day	mountainday
'Mountain Day' observed	mountaindayobserved
Respect for the Aged Day	respectfortheagedday
Autumn Equinox	autumnequinox
Health and Sports Day	healthandsportsday
Enthronement Ceremony Day	enthronementceremonyday
Culture Day	cultureday
'Culture Day' Observed	culturedayobserved
7-5-3 Day	753day
Labor Thanksgiving Day	laborthanksgivingday
Christmas	christmas
December 31 Bank Holiday	december31bankholiday
Emperor's Birthday	emperorsbirthday
'Emperor's Birthday' observed	emperorsbirthdayobserved
'Constitution Memorial Day' observed	constitutionmemorialdayobserved
Sports Day	sportsday
'New Year's Day' observed	newyearsdayobserved
'National Foundation Day' observed	nationalfoundationdayobserved
'Autumn Equinox' observed	autumnequinoxobserved
'Greenery Day' observed	greenerydayobserved
'Labor Thanksgiving Day' observed	laborthanksgivingdayobserved
Bridge Public holiday	bridgepublicholiday
'Spring Equinox' observed	springequinoxobserved
December 31 Bank Holiday	december31bankholiday

Japanese holidays

Holidays (ko-kr)

Holiday name	String to use to check for a festival match
New Year's Day	newyearsday
Independence Movement Day	independencemovementday
Labor Day	laborday
Children's Day	childrensday
Parents' Day	parentsday
Teacher's Day	teachersday
Buddha's Birthday	buddhasbirthday
Memorial Day	memorialday
Constitution Day	constitutionmemorialday
Liberation Day	liberationday
Chuseok	chuseok
Armed Forces Day	armedforcesday
National Foundation Day	nationalfoundationday
Hangeul Proclamation Day	hangeulproclamationday
Christmas Eve	christmaseve
Christmas Day	christmas
New Year's Eve	newyearseve
National Assembly Election Day	nationalassemblyelectionday

Korean holidays

Currency support

The `@sys-currency` system entity recognizes different currencies in different geographic regions.

The following sections list the currencies that `@sys-currency` recognizes for different locales.

Currencies (en-us)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
CAD	CA$
AUS	A$
GBP	£,gbp,quid

INR	₹,inr,rs,rs.
EUR	€,eur
JPY	¥
MXN	mxn
CHF	chf

English currencies

Currencies (cs-cz)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥,jpy
CZK	kc,kč,czk
MXN	mxn

Czech currencies

Currencies (da-dk)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn
DKK	dkk
SEK	sek
NOK	nok

Czech currencies

Currencies (nl-nl)

Currency code	Example of recognized values (case insensitive)

USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn

Dutch currencies

Currencies (fr-fr)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn
CHF	fr.,chf

French currencies

Currencies (de-de)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn
CHF	chf

German currencies

Currencies (it-it)

Currency code	Example of recognized values (case insensitive)
USD	$,usd,us$

GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn

Italian currencies

Currencies (pt-pt)

Currency code	Example of recognized values (case insensitive)
USD	$,usd,us$
BRL	r$,brl
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn

Portuguese currencies

Currencies (sk-sk)

Currency code	Example of recognized values (case insensitive)
USD	$
GBP	£
INR	₹
EUR	€
JPY	¥
CZK	kc,kč

Slovak currencies

Currencies (es-es)

Currency code	Example of recognized values (case insensitive)
USD	$,usd,us$
GBP	£,gbp
INR	₹,inr

EUR	€,eur
JPY	¥
MXN	mxn

Spanish currencies

Currencies (sv-se)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn
DKK	dkk
SEK	sek
NOK	nok

Swedish currencies

Currencies (tr-TR)

Currency code	Example of recognized values (case insensitive)
TRL	₺,tl,try
USD	$,usd
GBP	£,gbp,quid
INR	₹,inr,rs
EUR	€,eur
JPY	¥
MXN	mxn

Turkish currencies

Currencies (ar-ar)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
GBP	£,gbp

INR	₹,inr
EUR	€,eur
JPY	¥
MXN	mxn
EGP	E£,egp
IQD	iqd
SYP	syp

Arabic currencies

Currencies (iw-il)

Currency code	Example of recognized values (case insensitive)
USD	$
GBP	£
INR	₹
EUR	€
JPY	¥
MXN	mxn
ILS	שקל

Hebrew currencies

Currencies (zh-cn)

Currency code	Example of recognized values (case insensitive)
USD	$
GBP	£
INR	₹
EUR	€
CNY	¥

Simplified Chinese currencies

Currencies (zh-tw)

Currency code	Example of recognized values (case insensitive)
USD	$,usd,美金
GBP	£

INR	₹
EUR	€,歐元
CNY	¥,人民幣
TWD	nt$,twd,台幣
HKD	港幣 $

Traditional Chinese currencies

Currencies (ja-jp)

Currency code	Example of recognized values (case insensitive)
USD	$,usd,米ドル,usドル
GBP	£,gbp
INR	₹,inr
EUR	€,eur
JPY	¥,jpy
MXN	mxn

Japanese currencies

Currencies (ko-kr)

Currency code	Example of recognized values (case insensitive)
USD	$
GBP	£
INR	₹
EUR	€
JPY	¥
KRW	₩

Korean currencies

Currencies (universal model)

Currency code	Example of recognized values (case insensitive)
USD	$,usd
CAD	$,c$,ca$,can$,cad
AUD	a$,au$,aud
CHF	chf

GBP	£,gbp,quid
INR	₹,inr,rs,rs.
EUR	€,eur
JPY	¥,jpy
MXN	mxn
RUB	₽,rub

Universal model currencies

FAQs for watsonx Assistant

Find answers to frequently-asked questions and quick fixes for common problems.

FAQs about watsonx Assistant

What is watsonx Assistant?

IBM® watsonx™ Assistant is an improved way to build, publish, and improve virtual assistants. You use actions to build conversations. Actions are a simple way for anyone to create assistants. For more information, see the Getting Started guide or the documentation.

Why can't I see the assistants that I made with the classic experience in watsonx Assistant?

IBM® watsonx™ Assistant is a clean slate in the same IBM Cloud instance as your classic experience. Assistants that you created in one experience don't appear in the other. However, you can switch back and forth between experiences without losing any work. For more information, see Switching between watsonx Assistant and the classic experience.

What happens when I switch between the classic experience and watsonx Assistant?

The assistants that you create in one experience don't transfer to the other. However, you can switch experiences, return to your work, and create or use assistants. You don't lose anything by switching. Changing experiences doesn't affect other users in the same instance. For more information, see Switching between watsonx Assistant and the classic experience.

Is the classic experience still available?

IBM has no plans to discontinue the classic experience. However, we encourage you to explore the benefits and capabilities in watsonx Assistant. For more information, see the Getting Started guide. You can also continue to use dialog in watsonx Assistant. For more information, see Migrating to watsonx Assistant.

Where are the search skill and channel integrations in watsonx Assistant?

In the left navigation, click **Integrations** . On the Integrations page, you can add search, channel, and extension integrations to your assistant. For more information, see Adding integrations.

Where is the Assistant ID found in the new product experience?

The assistant ID can be found in **Assistant settings**.

In **Assistant settings**, the assistant ID is in the **Assistant IDs and API details** section.

What do the draft and live tags mean?

A `Draft` tag indicates that the information is linked to your draft environment, which means that you can preview these updates but they are not visible to your users. A `Live` tag indicates that the information is linked to your live environment, which means that the content is available to your users to interact with.

For more information, see Environments.

Why can't I log in?

If you can't log in to a service instance or see messages about tokens, such as `unable to fetch access token` or `400 bad request - header or cookie too large`, it might mean that you need to clear your browser cache. Open a private browser window, and then try again.

- If the private browsing window fixes the issue, then consider always using a private window or clear the cache of your browser. You can typically find an option for clearing the cache or deleting cookies in the browser's privacy and security settings.
- If the private browsing window doesn't fix the issue, then try deleting the API key for the instance and creating a new one.

Why am I being asked to log in repeatedly?

If you keep getting messages, such as `you are getting redirected to login`, it might be due to one of the following things:

- The Lite plan that you were using expired. Lite plans expire if they are not used within a 30-day span. To begin again, log in to IBM Cloud and create a new service instance of watsonx Assistant.

- An instance is locked when you exceed the plan limits for the month. To log in successfully, wait until the start of the next month when the plan limit totals are reset.

Why don't I see the Analytics page?

To view the **Analytics** page, you must have a service role of Manager and a platform role of at least Viewer. For more information about access roles and how to request an access role change, see Managing access to resources.

Why am I unable to view the API details, API key, or service credentials?

If you cannot view the API details or service credentials, it is likely that you do not have Manager access to the service instance in which the resource was created. Only people with Manager access to the instance can use the service credentials.

Can I configure the timeout parameter for a custom extension?

No, the timeout value for a custom extension is not configurable. Any call to the external API must complete within 48 seconds.

Why can't I edit intents, entities, or dialog nodes?

To edit a dialog, you must have Writer service access to the service instance and a platform role of at least Viewer. For more information about access roles and how to request an access role change, see Managing access to resources.

Can I export the user conversations from the Analytics page?

You cannot directly export conversations from the conversation page. However, you can use the `/logs` API to list events from the transcripts of conversations that occurred between your users and your assistant. For more information, see the V2 API reference. Or, you can use a Python script to export logs. For more information, see export_logs_py.

Can I export and import dialog nodes?

No, you cannot export and import dialog nodes from the product user interface.

If you want to copy dialog nodes from one dialog into another dialog, follow these steps:

1. Download as JSON files both the dialog that you want to copy the dialog nodes from and the dialog that you want to copy the nodes to.
2. In a text editor, open the JSON file for the dialog that you want to copy the dialog nodes from.
3. Find the `dialog_nodes` array, and copy it.
4. In a text editor, open the JSON file for the dialog skill that you want to copy the dialog nodes to, and then paste the `dialog_nodes` array into it.
5. Import the JSON file that you edited in the previous step to create a new dialog skill with the dialog nodes you wanted.

Is it possible to recover a deleted dialog?

Regularly back up data to prevent problems that might arise from inadvertent deletions. If you do not have a backup, there is a short window of time during which a deleted dialog might be recoverable. Immediately following the deletion, open a case with Support to determine if the data can be recovered. Include the following information in your case:

- Skill ID
- Instance ID or name
- Region where the service instance is hosted from which the dialog was deleted

Can I change my plan to a Lite plan?

No, you cannot change from an Enterprise or a Plus plan to a Lite plan.

How many Lite plan instances of watsonx Assistant can I create?

You can have only one Lite plan instance of watsonx Assistant per resource group.

How long are log files kept?

The length of time for which messages are retained depends on your service plan. For more information, see Log limits.

How do I create a webhook?

To define a webhook and add its details, go to the **Live environment** page and open the **Environment settings** page. From the **Environment settings** page, click **Webhooks > Pre-message webhook**. You can add details about your webhook. For more information, see Making a call before processing a message.

Can I have more than one entry in the URL field for a webhook?

No, you can define only one webhook URL for an action. For more information, see Defining the webhook.

Can I extend the webhook time limit?

No. The service that you call from the webhook must return a response in 8 seconds or less, or the call is canceled. You cannot increase this time limit.

Is there a range of IP addresses that are being used by a webhook?

Unfortunately, the IP address ranges from which watsonx Assistant might call a webhook URL are subject to change, which in turn prevent using them in any static firewall configuration. Use the https transport and specify an authorization header to control access to the webhook.

Why did I receive the message “Query cancelled” when I import a dialog?

This message is displayed when the dialog import stops because artifacts in the dialog, such as dialog nodes or synonyms, exceed the plan limits.

If a timeout occurs due to the size of the dialog but no plan limits are exceeded, you can reduce the number of elements that are imported:

1. Make a copy of the JSON file that you are trying to import.
2. Open the copy of the JSON file in an editor, and delete the `entities` array.
3. Import the edited JSON file as a new skill.
4. If this step is successful, edit the original copy of the JSON file.
5. Remove the `dialog_nodes`, `intents`, and `counterexamples` arrays.
6. Update the skill by using the API. Be sure to include the workspace ID and the `append=true` flag, as in this example:

```
curl -X POST -H "content-type: application/json" -H "accept: application/json" -u "apikey:{apikey}" -d@./skill.json "url/api/v1/workspaces/{workspace_id}?version=2019-02-28&append=true"
```

What do I do if the training process seems stuck?

If the training process gets stuck, first check whether for an outage for the service by going to the Cloud status page. You can start a new training process to stop the current process and start over.

How do I see my monthly active users in watsonx Assistant?

To see your monthly active users (MAU):

1. Sign in to https://cloud.ibm.com
2. Click the **Manage** menu, then choose **Billing and usage**.
3. Click **Usage**.
4. For watsonx Assistant, select **View Plans**.
5. Under **Time Frame**, select the month that you need.
6. Select your Plus plans or Plus Trial plans to see monthly active users and the API calls.

Error: New Off Topic not supported

You see the error `New Off Topic not supported` after you edit the JSON file for a dialog and changing the skill language from English to another language.

To resolve this issue, modify the JSON file by setting `off_topic` to `false`. For more information about this feature, see Defining what's irrelevant.

Can I see what web browser users are using with watsonx Assistant?

With the V2 API and an Enterprise plan, you can use the Segment extension to see what browser was used to send the message. For more information, see Sending events to Segment.

Is it possible to increase the number of intents per skill

No, it is not possible to increase the number of intents per skill.

Getting help

Get help with solving issues that you encounter while using the product.

Use these resources to get answers to your questions:

- For answers to frequently asked questions, see the FAQ.
- Find answers to common questions or ask questions where experts and other community members can answer by visiting the watsonx Assistant Community forum.

If your service plan covers it, you can get help by creating a case from IBM Cloud Support.

Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

IBM® watsonx™ Assistant uses standard Windows navigation keys.

Accessibility features in the product documentation

Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or with other special needs, use information technology products successfully.

The accessibility features in this product documentation allow users to do the following:

- Use screen-reader software and digital speech synthesizers to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using assistive technologies with HTML-based information.
- Use screen magnifiers to magnify what is displayed on the screen.
- Operate specific or equivalent features by using only the keyboard.

The documentation content is published in the IBM Cloud Docs site. For information about the accessibility of the site, see Accessibility features for IBM Cloud.

Browser support

The watsonx Assistant application requires the same level of browser software as is required by IBM Cloud. For more information, see IBM Cloud prerequisites.

For information about the web browsers that are supported by the web chat integration, see Browser support.

Terms and notices

See IBM Cloud Terms and Notices for information about the terms of service.

US Health Insurance Portability and Accountability Act (HIPAA) support is available with *Enterprise with Data Isolation* plans that are hosted in the Washington, DC location created on or after 1 April 2019. For more information, see Enabling HIPAA support for your account.

To learn more about service terms and data security, read the following information:

- Service terms (Search for the watsonx Assistant offering)
- Data Processing and Protection Datasheet
- IBM Cloud Data security and privacy

IBM Corporation
New Orchard Road
Armonk, NY 10504

Produced in the United States of America
2025-10-21