RAG as watsonx Orchestrate AI Assistant extension

Table of Contents

Introduction

This document is a guideline for using the RAG Accelerator deployment with watsonx Orchestrate AI Assistant Builder to implement a question answering service. The resulting AI Assistant calls the deployed RAG function endpoint from the watsonx Runtime deployment space.

Additionally there is also option for the users to configure feedback logging based on the answers returned by the RAG function. There is also an option for users to search and retrieve expert profile contact information to reach out any further information regarding the Q&A is necessary.

Remark: This is a guideline to setup an AI Assistant with watsonx Orchestrate. However, everything described here applies to IBM watsonx Assistant as well.

Prerequisites

- watsonx Orchestrate service has been instantiated on IBM Cloud or Cloud Pak for Data.
- The RAG Accelerator python function is deployed and running in a deployment space. The deployment space is either available on IBM Cloud or watsonx.ai software (on-premise version of watsonx.ai).
- The Serving Name is known. It is configured as parameter `deployment_serving_name` in the *Q&A with RAG Accelerator* project. Alternatively, find the serving name in the deployment information box by navigating to ***Deployments*** → ***Spaces*** → ***your deployment space*** → ***your deployment***. Click on ***i*** as per image below for deployment information.

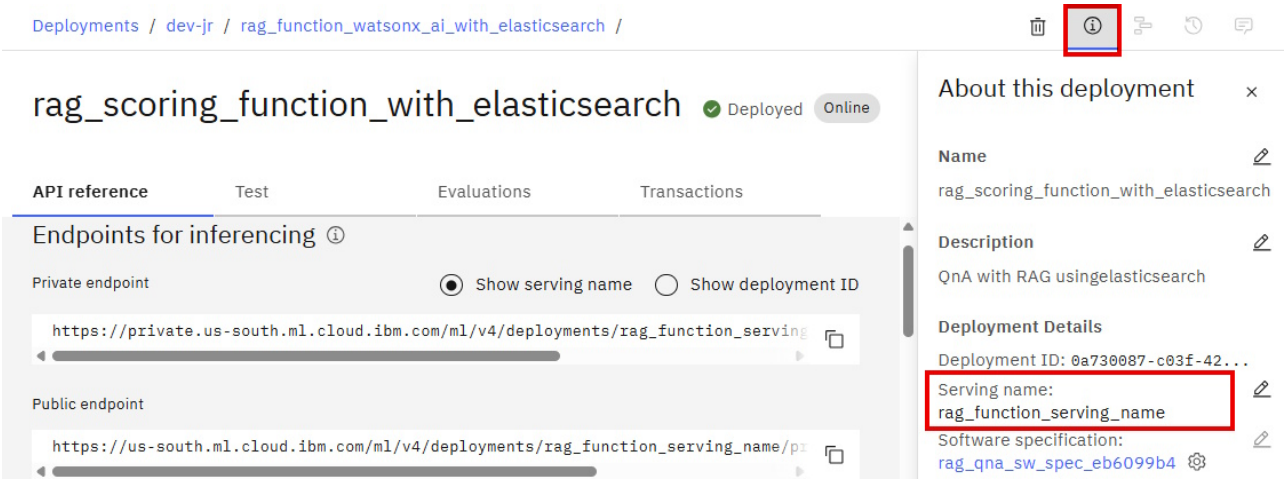


Figure: watsonx Runtime deployment serving name

- If the RAG function is deployed on **IBM Cloud**, an IAM API key is needed. Under the hood, watsonx Assistant will use it to generate an access token.
- If the RAG function is deployed on **watsonx.ai software**, a ZenApiKey or a bearer token or user id and password / API key, respectively, are needed. Find details here.

Create a new AI Assistant

Launch **watsonx Orchestrate AI Assistant Builder**, expand dropdown list of available assistants and select ***+ Create New***. Enter an ***Assistant name*** and a ***Description***. Also, choose ***Assistant language*** from dropdown list. Click ***Create assistant***.

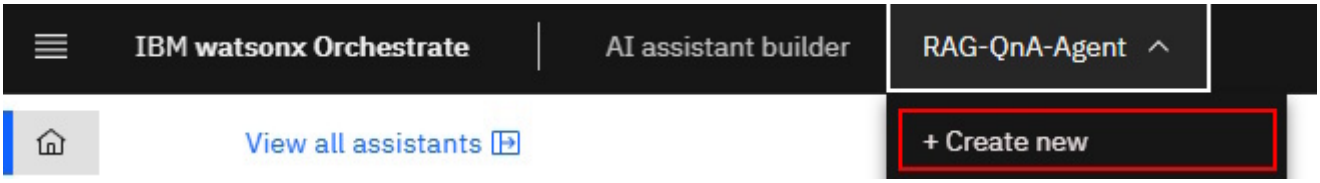


Figure: Create new assistant

Setup a Custom Extension using the OpenAPI specification for RAG deployment

To make the RAG function to available to AI Assistants, a custom extension must be configured and added.

Download and apply OpenAPI specification file

The RAG function is being integrated into watsonx Orchestrate AI Assistant by an extension that must be built from the corresponding OpenAPI specification.

The RAG Accelerator project contains the OpenAPI specification as data asset. Proceed as follows to download the file to your local computer. In the watsonx.ai project, click tab ***Assets***, select ***Data*** for asset type, click the vertical ellipsis menu for data asset ***RAG-OpenAPI.json*** and select ***Download*** from the menu.

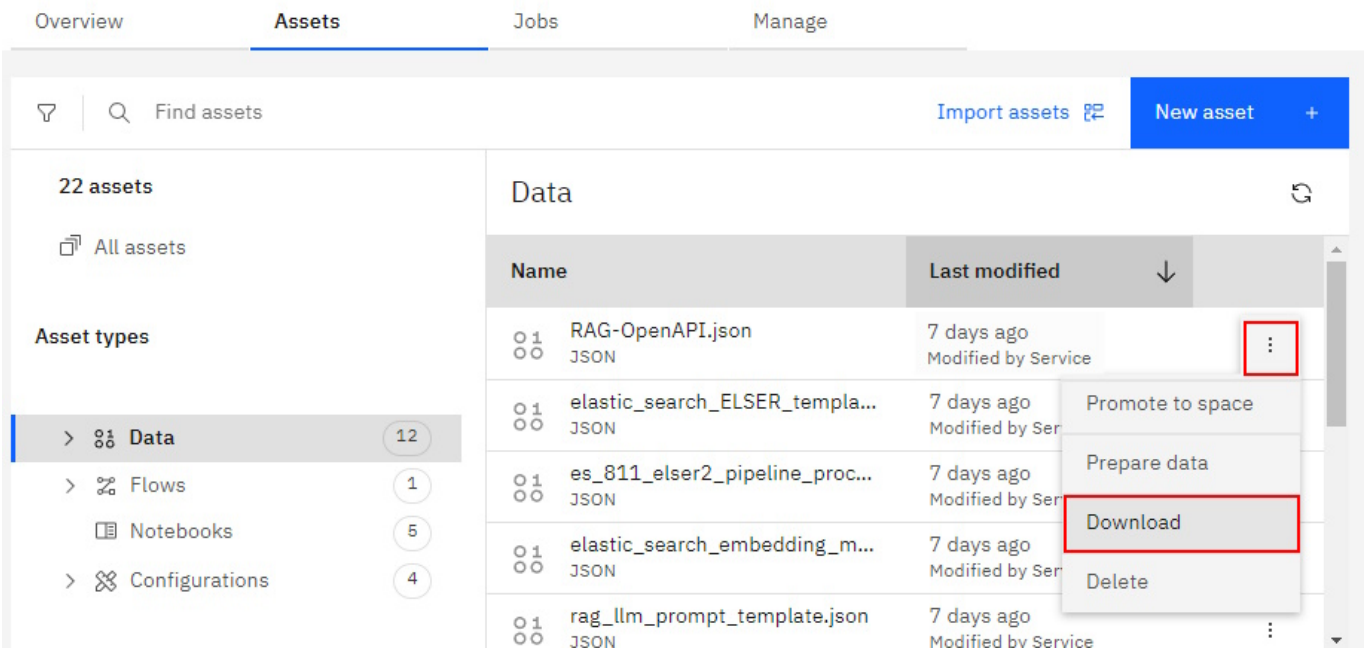


Figure: Data Assets

The downloaded OpenAPI specification file contains all fragments mentioned below, except for the watsonx.ai software server name. It must be adapted if the RAG function is not deployed on IBM Cloud.

Alternatively, create the OpenAPI specification for the deployed RAG function in JSON format. Depending on where the RAG functions is deployed, the OpenAPI specification must include key `components/securitySchemes/oAuth2` for IBM Cloud IAM access or key `components/securitySchemes/basicAuth` or `.../bearerAuth` or `.../apiKeyAuth`, respectively, for access to watsonx.ai software.

```
{
    "components": {
        "securitySchemes": {
            "oAuth2": {
                "description": "IAM access (token)",
                "type": "oauth2",
                "flows": {
                    "x-apikey": {
                        "tokenUrl":
"https://iam.cloud.ibm.com/identity/token",
                        "grantType": "urn:ibm:params:oauth:grant-
type:apikey",
                        "secretKeys": [ "apikey" ],
                        "paramKeys": [],
                        "scopes": {}
                    }
                }
            },
            "basicAuth": {
                "description": "User id and password/user api key",
                "type": "http",
                "scheme": "basic"
            },
```

```
            "bearerAuth": {
                "description": "Bearer token",
                "type": "http",
                "scheme": "bearer"
            },
            "apiKeyAuth": {
                "description": "ZenApiKey",
                "type": "apiKey",
                "in": "header",
                "name": "Authorization"
            }
        },
    }
}
```

Furthermore, watsonx Orchestrate AI Assistant requires server information as part of the OpenAPI specification. Therefore, add key `servers` and include the API endpoint URL of your watsonx.ai Runtime to that list, see watsonx.ai Runtime API reference.

```
{
    "servers": [
        {
            "description": "watsonx.ai Runtime API Endpoint - IBM Cloud
(Dallas)",
            "url": "https://us-south.ml.cloud.ibm.com"
        },
        {
            "description": "watsonx.ai Runtime API Endpoint - watsonx.ai
installed",
            "url": "<Endpoint URL of your WML service>"
        }
    }
    ]
}
```

Add RAG extension to catalog

Launch **watsonx Orchestrate AI Assistant Builder** and follow the path below to add a custom RAG extension to the catalog.

- Click ***Integrations*** on navigation pane
- Click ***Build custom extension*** button
- Tab ***Get started***: Click ***Next***
- Tab ***Basic information***: Enter ***Extension name*** and ***Extension description***, click ***Next***
- Tab ***Import OpenAPI***: Drag and drop or browser to the `OpenAPI \*.json file`. When loaded, click ***Next***

- Tab ***Review extension***: Click ***Finish***

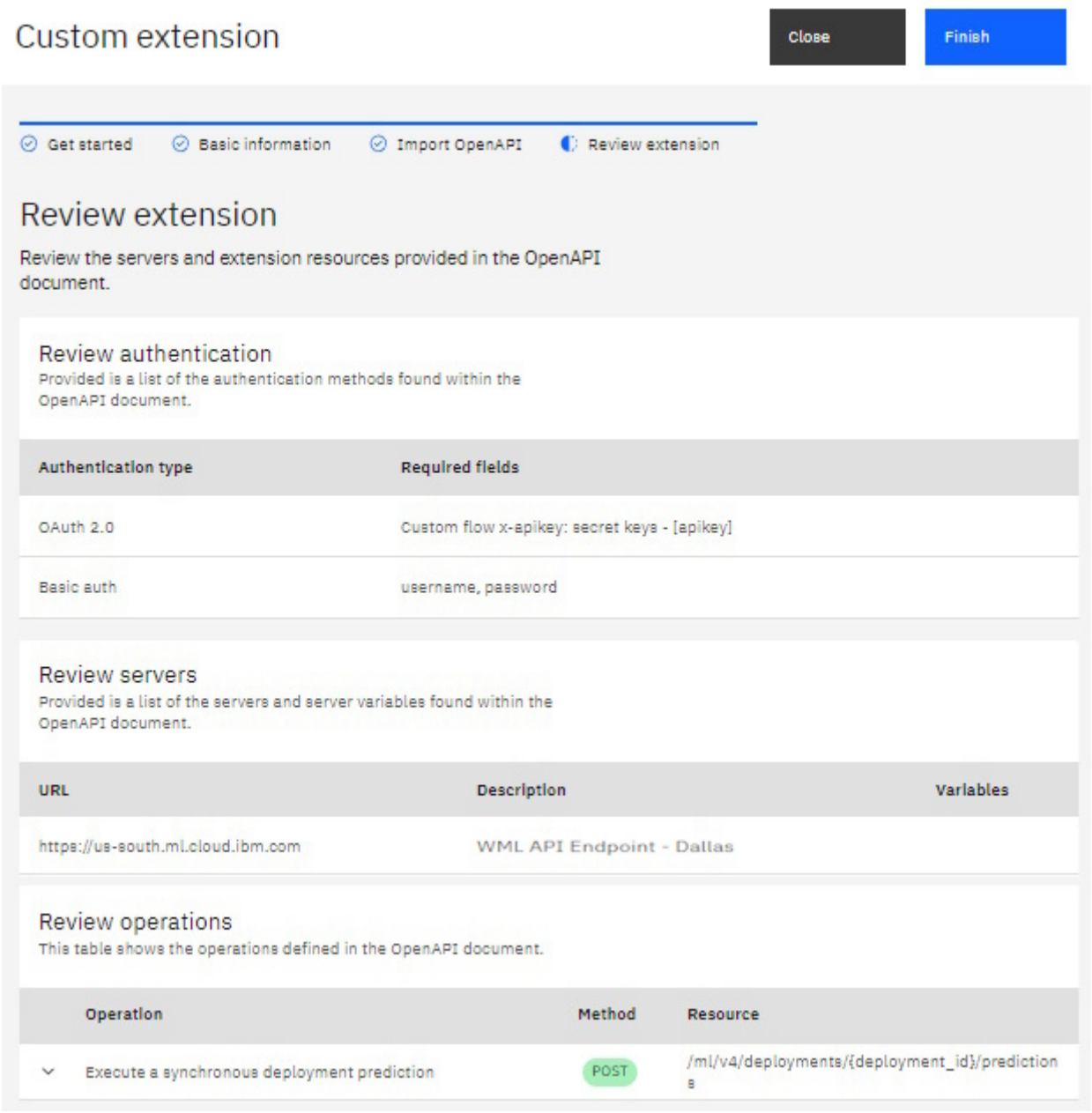


Figure: RAG extension

Add RAG extension to watsonx Orchestrate AI Assistant

A new tile appears that represents the RAG extension. In order to use the RAG extension with your assistant, proceed as follows:

- Click ***Add*** on the RAG extension tile and ***Add*** again on the popup dialog.
- Tab ***Get started***: Click ***Next***
- Tab ***Authentication***:
    - If RAG function is deployed on **IBM Cloud**, enter:
      Authentification Type: `OAuth 2.0`
      Grant type: `Custom apikey`
      Apikey: `<your IAM API Key>`
      Client authentication: `Send as Body`

Header prefix: `Bearer`
Servers: `<Endpoint URL of your WML service>`

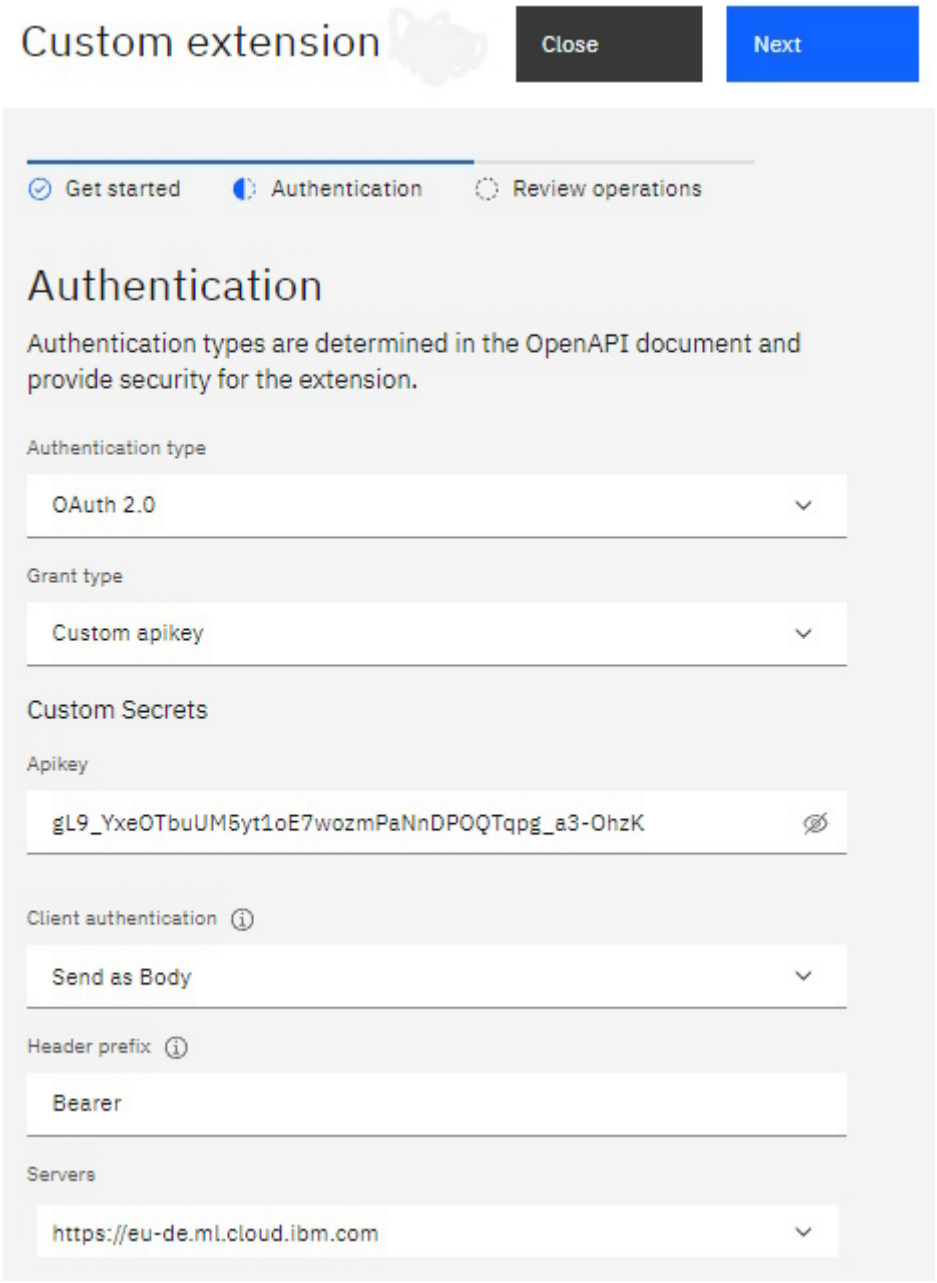


Figure: RAG extension authentication (IBM Cloud)

- If RAG function is deployed on **watsonx.ai software**:
  There are multiple options to authenticate the RAG extension, see here. One option is to generate a ZenApiKey as described here and filling out the form as follows:
  Authentification Type: `API key auth`
  API key: `ZenApiKey <USER:APIKEY, base64 encoded>`
  Servers: `<Endpoint URL of your WML service>`

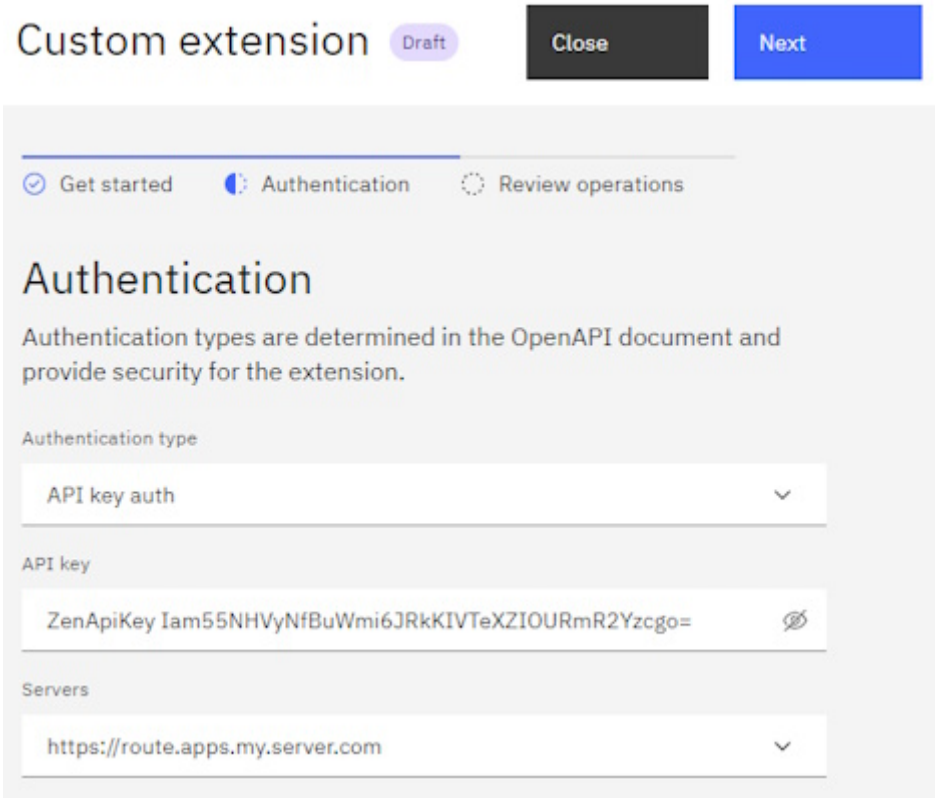


Figure: RAG extension authentication (watsonx.ai software)

Click ***Next***

- Tab ***Review operations***: Click ***Finish***

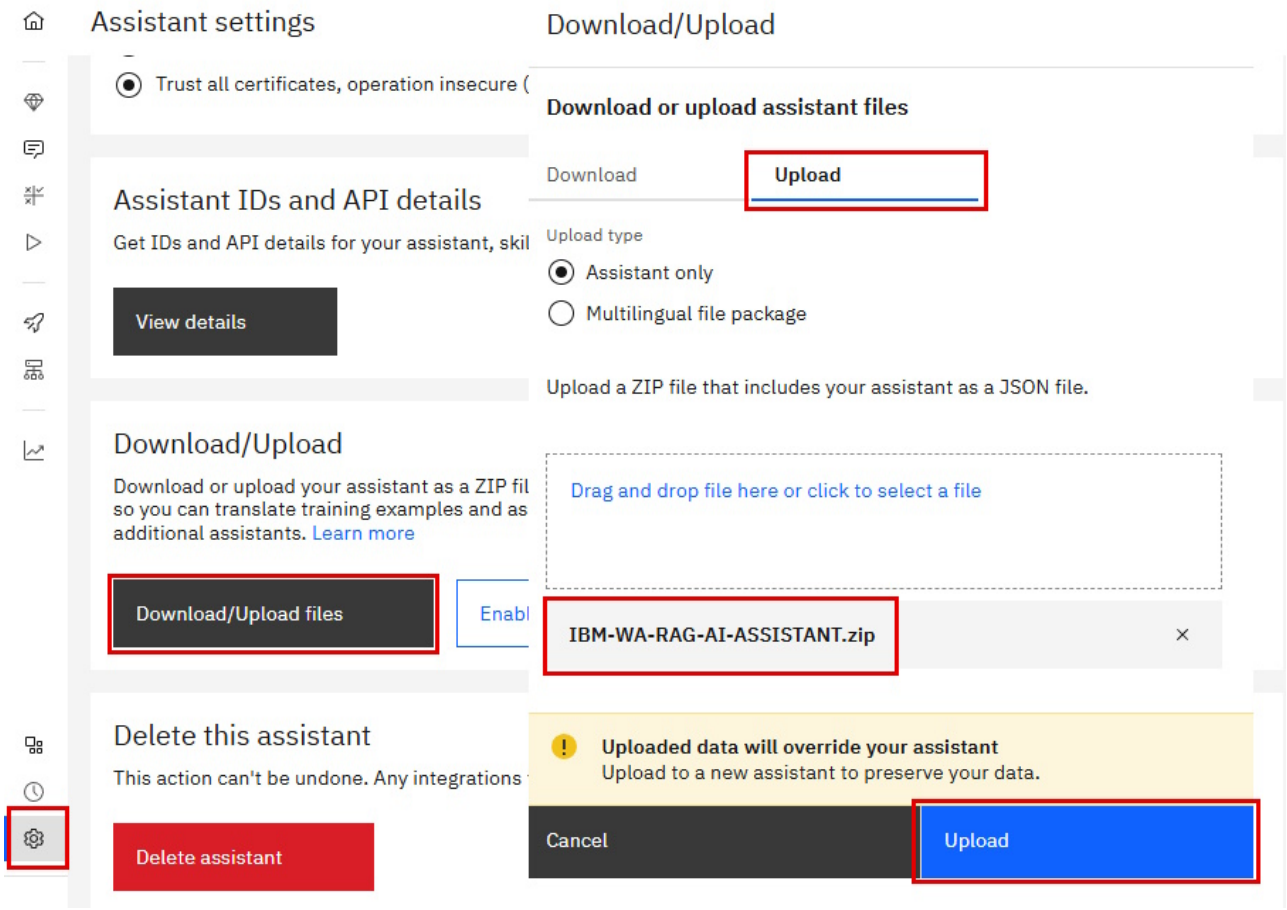
The RAG extension is now ready to be used in an action skill.

Upload AI Assistant sample files

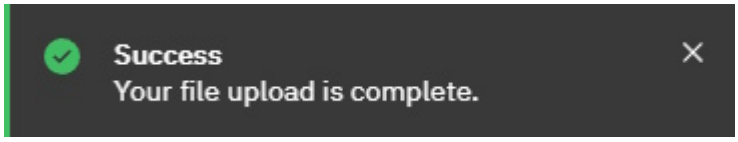
The following sections give step-by-step instructions to implement an AI Assistant that is powered by the RAG function. As an alternative to performing all these steps manually, you can upload the AI Assistant actions from the *Q&A with RAG Accelerator* project. Proceed as follows.

1. In the **watsonx.ai project**, click tab ***Assets***, select ***Data*** for asset type, click the vertical ellipsis menu for data asset `IBM-WA-RAG-AI-ASSISTANT.zip` and select ***Download*** from the menu.

2. In **watsonx Orchestrate AI Assistant Builder**, select ***Assistant settings*** → ***Download/Upload***, select tab ***Upload***, drop file `IBM-WA-RAG-AI-ASSISTANT.zip` into the provided field, click ***Upload*** and confirm by clicking ***Upload and replace***.



3. Wait for success message. Afterwards the uploaded AI Assistant is available.



Instructions to setup the watsonx Orchestrate AI Assistant

Steps to setup watsonx Orchestrate AI Assistant is as follows:

Create session variable for storing the Deployment Serving Name

From tool bar, navigate to ***Actions*** → ***Variables*** → ***Created by you***, and click ***New variable***. Enter `DeploymentServingName` for name, select type `Free Text`. For initial value, enter the serving name (text), see above. Click ***Save***.

Using the RAG extension in an action skill requires these steps:

1. Take the user query.
2. Call the RAG extension
3. Return the RAG extension response

Add Step 1

Use the tool bar on the left side to navigate to ***Actions***. Click ***New action***, and then ***Start from stratch***. Specify a phrase that triggers the new action, for example `help`, and click ***Save***. In field ***Assistant says*** enter a text that prompts the user to ask a question, for example `How can I help you?`. You also must

Define a customer response that enables the user to enter **free text** by selecting the corresponding item from the drop-down list and click ***Save***.

Add Step 2

Click ***New step***. Select ***Use an extension*** under the ***And then*** section. You need to specify the name of the RAG extension in field ***Extension*** and the method name in field ***Operation***, i.e. ***Execute a synchronous deployment prediction for QnA RAG***. Under ***Parameters*** set values based on the following table.

Parameter	Value
version	watsonx Runtime API version (text), e.g. `2021-05-01`
deployment_id	`${DeploymentServingName}` (expression), where `${DeploymentServingName}` is session variable
question	`${step_<1>}` (expression), where `${<step<1>}` is action step 1 variable

While entering the value for `${step_<1>}`, a context menu appears after you have entered the dollar-sign **($)**. (Note : You should have a blank between the squared brackets and the dollar-sign.) Select menu item ***Action step variables*** → ***1. ...***.

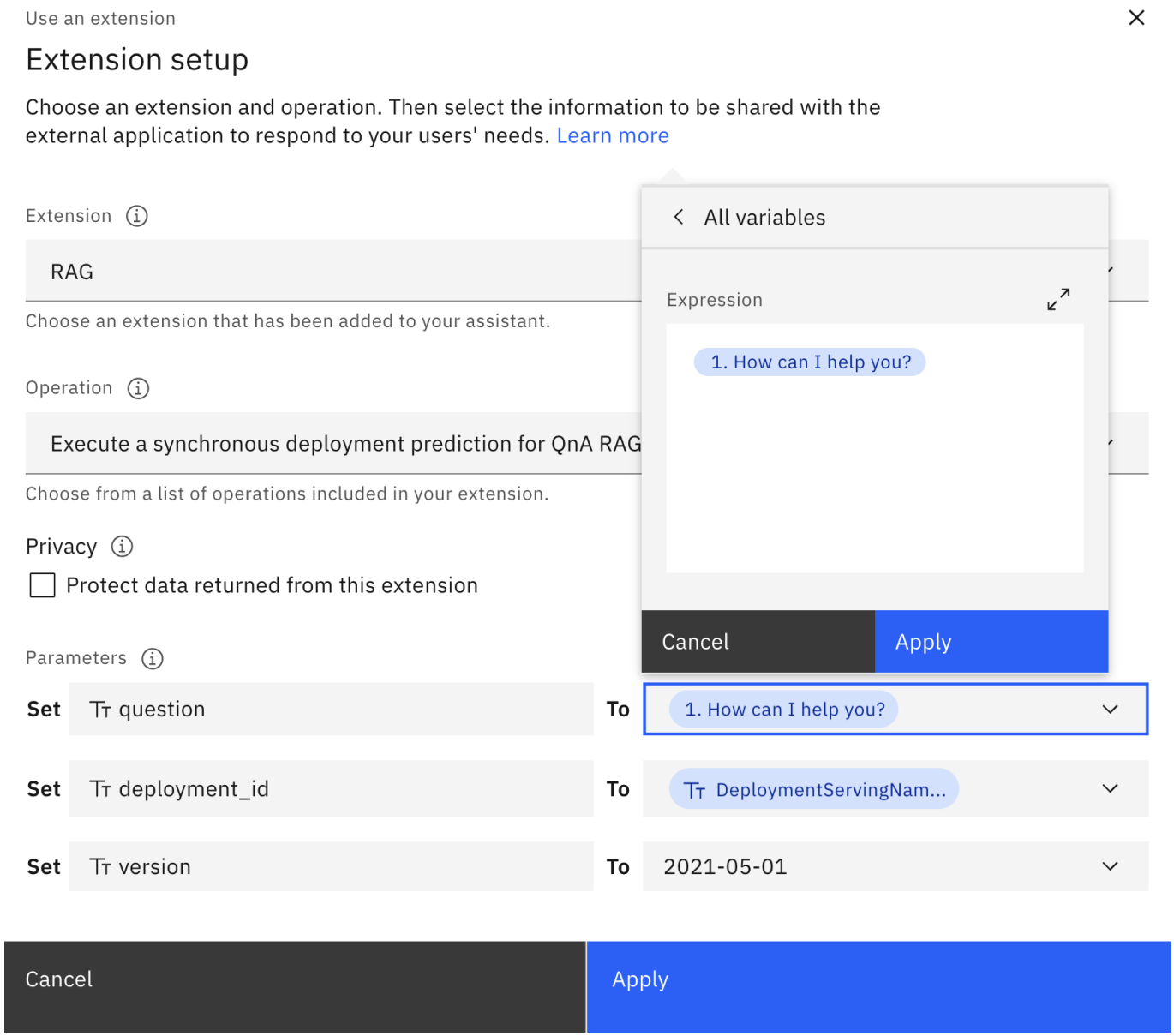


Figure: RAG extension setup

After values have been assigned to all parameters, click button ***Apply*** and click ***Save***. If the button does not become active it might be necessary to review the default value of parameter ***version*** and save it (without changes).

Add Step 3

Click ***New step***. Click ***Set variable value***, then ***Set new value*** and select ***New session variable***. Enter name of the new variable, i.e. `Answer`, choose type *Free text* and click ***Apply***. Assign the following expression to it (field ***To***):

```
${step_<2>_result_1.result}.response
```

where `${step_<2>_result_1.result}` is the RAG extension response variable. (After typing the dollar-sign, a context menu appears. Select `<RAG extension name> (step 2)` and complete the expression as shown above. Click ***Apply***

The same way assign the expression below to new session variable `References` to compile a list of refenerce web links.

```
${step_<2>_result_1.result}.source_documents_references.joinToArray("<a
target='_blank' href='%e.url%'>%e.title%</a>", true).join('<br/>')
```

Finally, put both new session variables in the field ***Assistant says***.

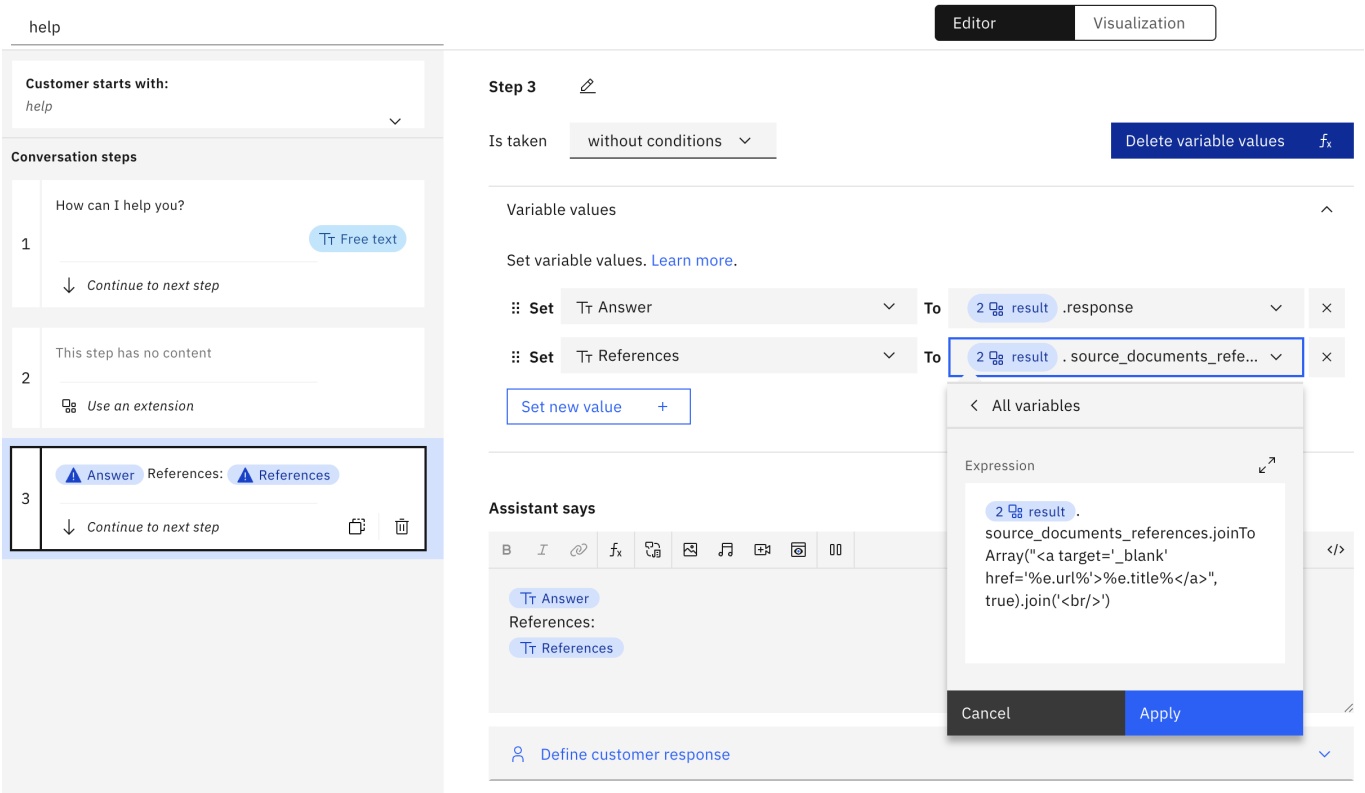


Figure: Action skill step for assistant response

That's it. When called, the action will post the user's input to the RAG function and return the response to the user. You can view it also via the **Preview** tab

Instructions to implement feedback feature with watsonx Orchestrate AI Assistant

If you have enables logging with the RAG function, you can add a feature to your watsonx Orchestrate AI Assistant that adds user's feedback to the corresponding log records. To do so, procced as follows.

Create session variable for storing the feedback information

From tool bar, navigate to ***Actions*** → ***Variables*** → ***Created by you***, and click ***New variable***. Enter `Feedback` for name, select type `Free Text`, and click ***Save***.

Repeat this procedure to create variables `FeedbackComment` and `LogID`.

Create action that sends the feedback

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, and click ***New action***, and the ***Start from stratch***. Do not enter an example (click ***Cancel***).
2. On the Untitled action field (input field at the left upper corner), enter action name `send_feedback`, save, click on **conversation steps 1** and select ***Use an extension*** under the ***And then*** section. You need to specify the name of the RAG extension in field ***Extension*** and the method name in field ***Operation***, i.e. ***Execute a synchronous deployment prediction for Feedback Logging***. For parameter log_id, value and feedback enter expression as below (and click ***Apply***).

Parameter	Value
log_id	`${LogID}` (expression), where `${LogID}` is session variable
value	`${Feedback}` (expression), where `${Feedback}` is session variable
comment	`${FeedbackComment}` (expression), where `${FeedbackComment}` is session variable

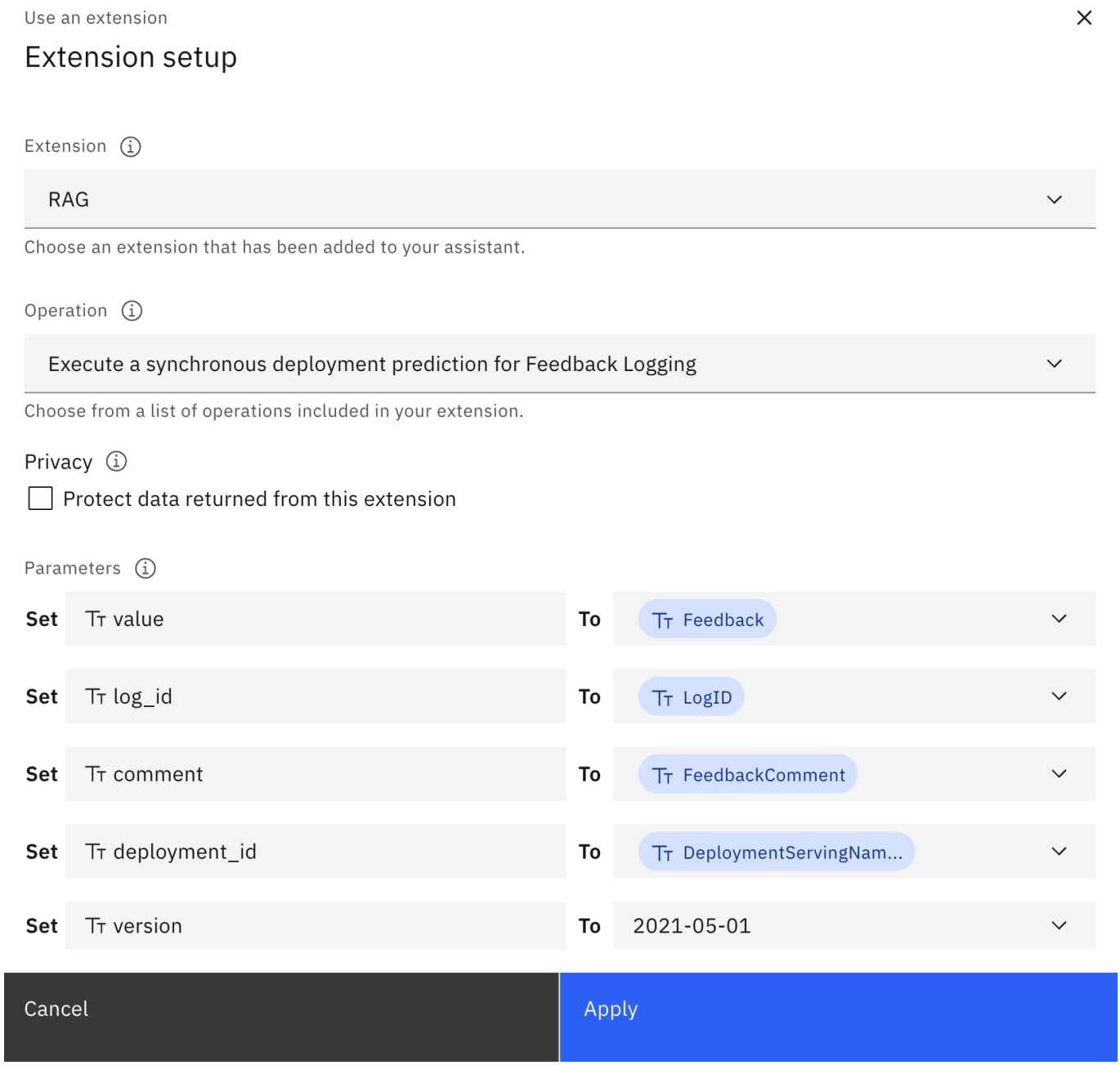


Figure: Extension parameters for sending feedback

3. Click ***New step***, enter `Your feedback has been sent.` in field ***Assistant says*** and select `End the action` under section ***And then*** and save.

Setup actions for feedback ratings options

Currently, we support both 2-star and 5-star feedback ratings. Please create actions that ask and receive feedback based on your rating options below to continue.

To Create action that asks for 2 Star feedback

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, and click ***New action***, and the ***Start from stratch***. Do not enter an example (click ***Cancel***). Enter action name `feedback` on the Untitled Action field (input field at the left upper corner), and click on conversation steps 1.
2. In section ***Assistant says*** switch to JSON editor by clicking the (***</>***) button and enter the JSON content below.

```
{ "generic": [
    { "response_type": "text",
      "values": [{"text_expression": {"concat": [{
                  "scalar": "Rate this answer."
                }]}}],
      "selection_policy": "sequential"
    },
    { "options": [
        { "label": "👍",
          "value": {"input": {"text": "👍"}}
        },
        { "label": "👎",
          "value": {"input": {"text": "👎"}}
        }
      ],
      "response_type": "option"
    }
  ]
}
```

3. Select `End the action` in section ***And then*** and save.

To Create action to receive 2 Star feedback

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, click ***New action***, and then ***Start from stratch***. Enter 👍 as an example (copy character from here and paste it). Click ***Save***.
2. Add `thumb up` to the action name (input field at the left upper corner), and click on conversation steps 1.
3. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `Feedback`, and click ***Enter text***. Type text `positive` and click ***Apply***.
4. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `FeedbackComment`, and click ***Enter text***. Type text `none` and click ***Apply***.
5. Repeat steps 1 to 3 for action 👎 , `thumb down` and `negative` feedback.

6. In section ***Asssitant says*** enter `Please leave a comment`. ***Define customer response*** as ***Free text***.
7. Click ***Next step +***.
8. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `FeedbackComment`, and click ***Action step variables*** and select ***1. Please leave a comment.***, click ***Apply***.
9. In section ***And then*** select ***Go to a subaction***. Select ***Go to*** `send_feedback` and click ***Apply***.

To Create action that asks for 5 Star feedback

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, and click ***New action***, and the ***Start from stratch***. Do not enter an example (click ***Cancel***). Enter action name `feedback` on the Untitled Action field (input field at the left upper corner), and click on conversation steps 1.
2. In section ***Assistant says*** switch to JSON editor by clicking the (***</>***) button and enter the JSON content below.

```
{
  "generic": [
    {
      "values": [
        {
          "text_expression": {
            "concat": [
              {
                "scalar": "Rate this answer."
              }
            ]
          }
        }
      ],
      "response_type": "text",
      "selection_policy": "sequential"
    },
    {
      "options": [
        {
          "label": "🤩 perfect",
          "value": {
            "input": {
              "text": "🤩"
            }
          }
        },
        {
          "label": "😀 good",
          "value": {
            "input": {
              "text": "😀"
            }
          }
        },
        {
```

```
          "label": "🙂 ok",
          "value": {
            "input": {
              "text": "🙂"
            }
          }
        },
        {
          "label": "😠 bad",
          "value": {
            "input": {
              "text": "😠"
            }
          }
        },
        {
          "label": "😡 very bad",
          "value": {
            "input": {
              "text": "😡"
            }
          }
        }
      ],
      "response_type": "option"
    }
  ]
}
```

3. Select `End the action` in section ***And then*** and save.

To Create action to receive 5 Star feedback

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, click ***New action***, and then
   Start from stratch. Enter 🤩 as an example (copy character from here and paste it). Click ***Save***.
2. Add 🤩 to the action name (input field at the left upper corner), and click on conversation steps 1.
3. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `Feedback`, and click ***Enter text***. Type text `perfect` and click ***Apply***.
4. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `FeedbackComment`, and click ***Enter text***. Type text `none` and click ***Apply***.
5. Repeat steps 1 to 3 for action 😀 and type text - `good`, 🙂 and type text - `ok`, 😠 and type text - `bad`, 😡 and type text - `very bad`, feedback
6. In section ***Asssitant says*** enter `Please leave a comment`. ***Define customer response*** as ***Free text***.
7. Click ***Next step +***.
8. Click ***Set variable value*** and then ***Set new value*** → ***Session variables*** → `FeedbackComment`, and click ***Action step variables*** and select ***1. Please leave a comment.***, click ***Apply***.
9. In section ***And then*** select ***Go to a subaction***. Select ***Go to*** `send_feedback` and click ***Apply***.

Call feedback action

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***. Open action `help` and click on conversation steps 3.
2. Click ***Set new value*** → ***Session variables*** → `LogID`, click ***Expression***. Enter the following expression:

```
${step_<2>_result_1.result}.log_id
```

where `${step_<2>_result_1.result}` is the RAG extension response variable. This step must be done similarly to the steps under the section **Add Step 3** section above.

3. In section ***And then*** select ***Go to a subaction***. Select ***Go to*** `feedback` and click ***Apply***.

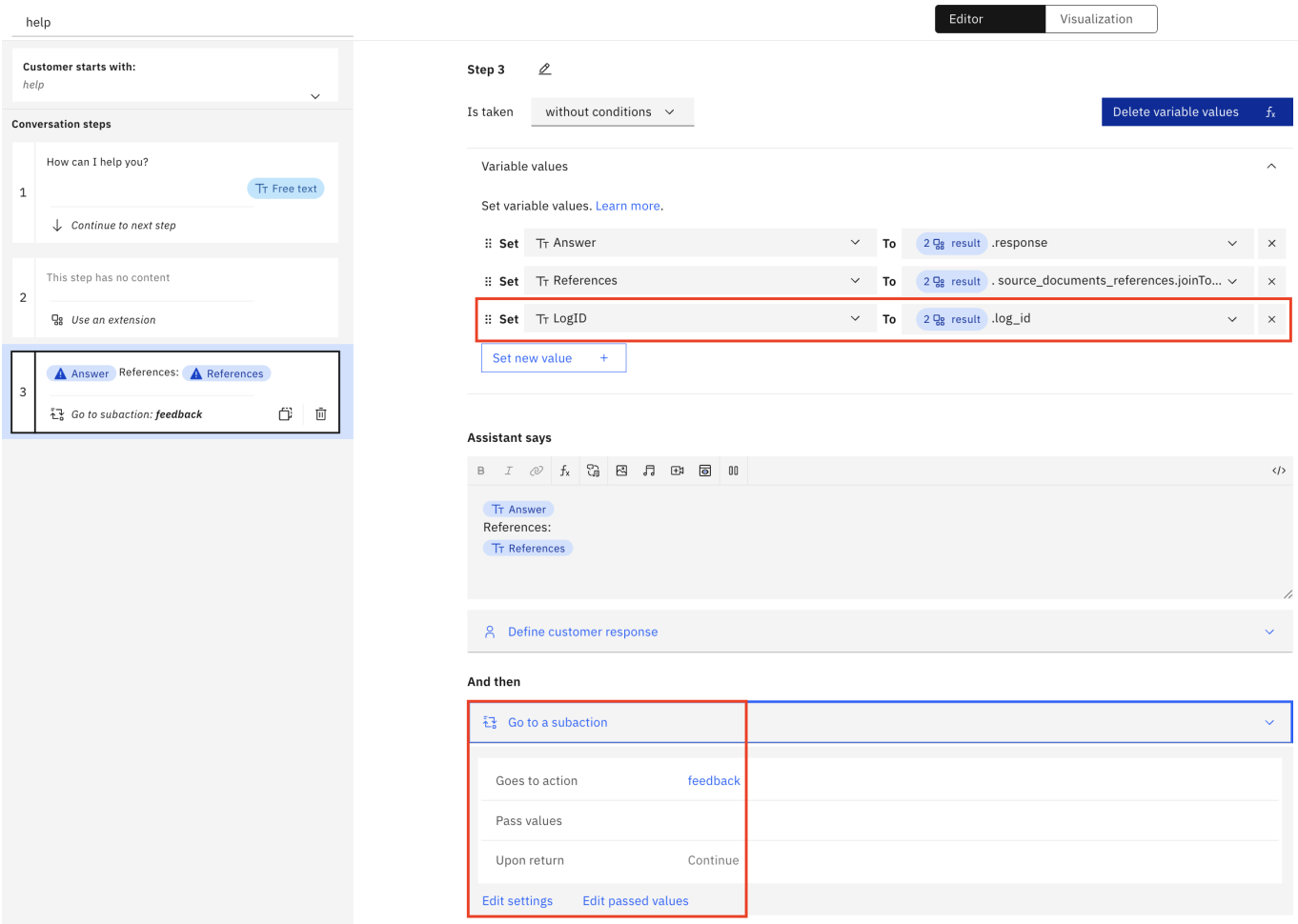


Figure: Updates applied to action `help`

Click Save.

RAG only watsonx Orchestrate AI Assistant

Action `help` that you have implemented is run when the end user explicitly asks for help. This is only one out of potential many actions that the AI Assistant is capable to perform. However, running RAG might be only AI Assistant's capability. In this case it might be good to run the RAG extension, i.e. the `help` action on

any user input. You can implement this by overriding the `No matches` action that have been set by the AI Assistant by default.

1. Navigate to ***Actions*** → ***Set by assistant*** → ***No matches***.
2. Delete all steps from this action.
3. Create a new step. In section ***And then*** select ***Go to a subaction***. Select ***Go to*** `help` and mark ***End this action after the other action is completed***. Click ***Apply***.
4. Click ***Edit passed values***. Click ***Set new value*** and select ***1. ...***. Enter the expression below and click ***Apply*** twice.

```
input.original_text ? input.original_text : input.text
```

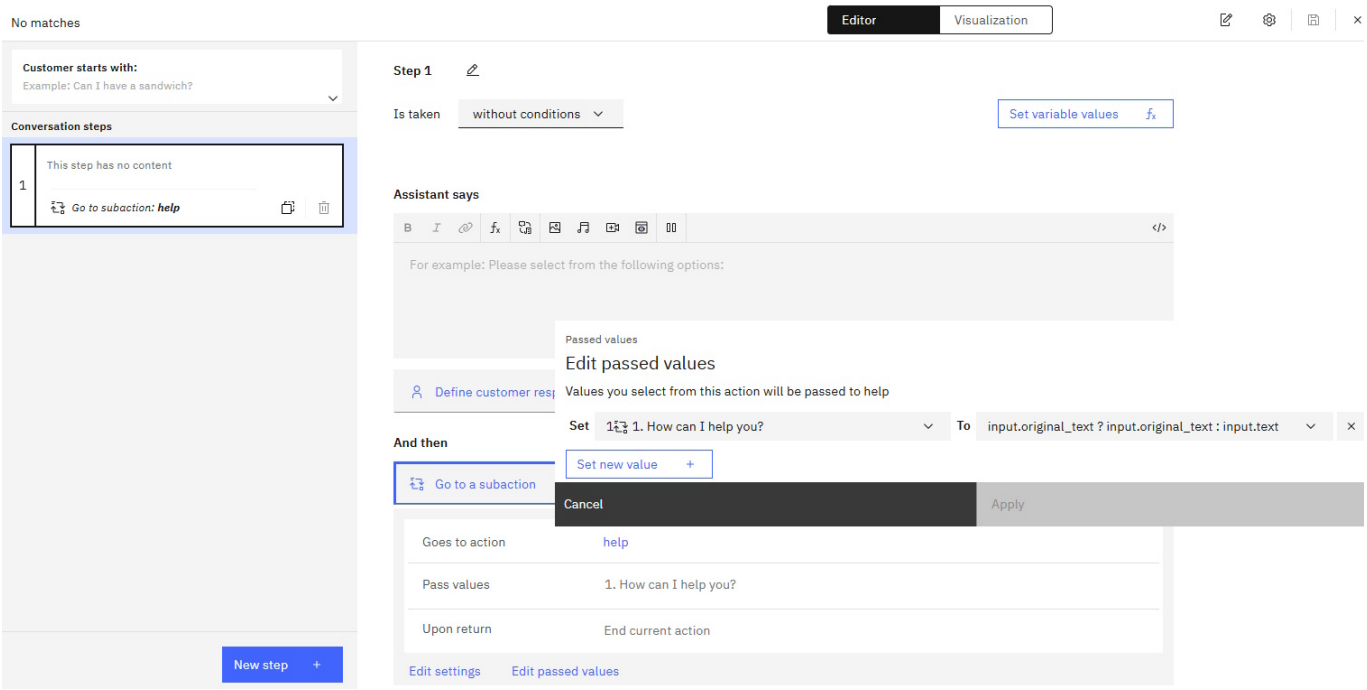


Figure: `No matches` action

Instructions to perform search of Expert Profile information with watsonx Orchestrate AI Assistant

If you have ingested the expert profile data into a subsequent vector index via Elastic/Milvus & have enabled this with the RAG function, you can add a feature to your watsonx Orchestrate AI Assistant skill. Regardless of what answer is being generated via the LLM inference, you will have the option to fetch an expert profile that would be recommended as a contact point for a given question/topic. To do so proceed as follows.

Create session variable for storing the expert profile result information

From the tool bar, navigate to ***Actions*** → ***Variables*** → ***Created by you***, and click **New variable**. Enter `ExpertResult` for name, select type `Any` and click ***Save***.

Repeat this procedure to create `ExpertEmail`, `ExpertID`, `ExpertName`, `Domain` and any other field you have ingested into the vector store via Elastic or Milvus. While creating the variables, under ***Type*** choose `Free Text`

Create action that fetches and displays the expert profile information

1. Navigate to ***Actions → All items → Created by you***, and click ***New action***, and the ***Custom-built action***.

2. On the Untitled action field (input field at the left upper corner), enter action name `get_expert_profile`, save. On the "Customer starts with" section, enter phrase `Get Expert Details`, save. Click on conversation steps 1 and select Use an extension under the ***And then*** section. You need to specify the name of the RAG extension in field ***Extension*** and the method name in field ***Operation***, i.e. ***Execute a synchronous deployment prediction for Expert Recommendation***. For parameter log_id, enter `${LogID}` as expression (and click Apply). Remaining values deployment_id and version will be same as before.

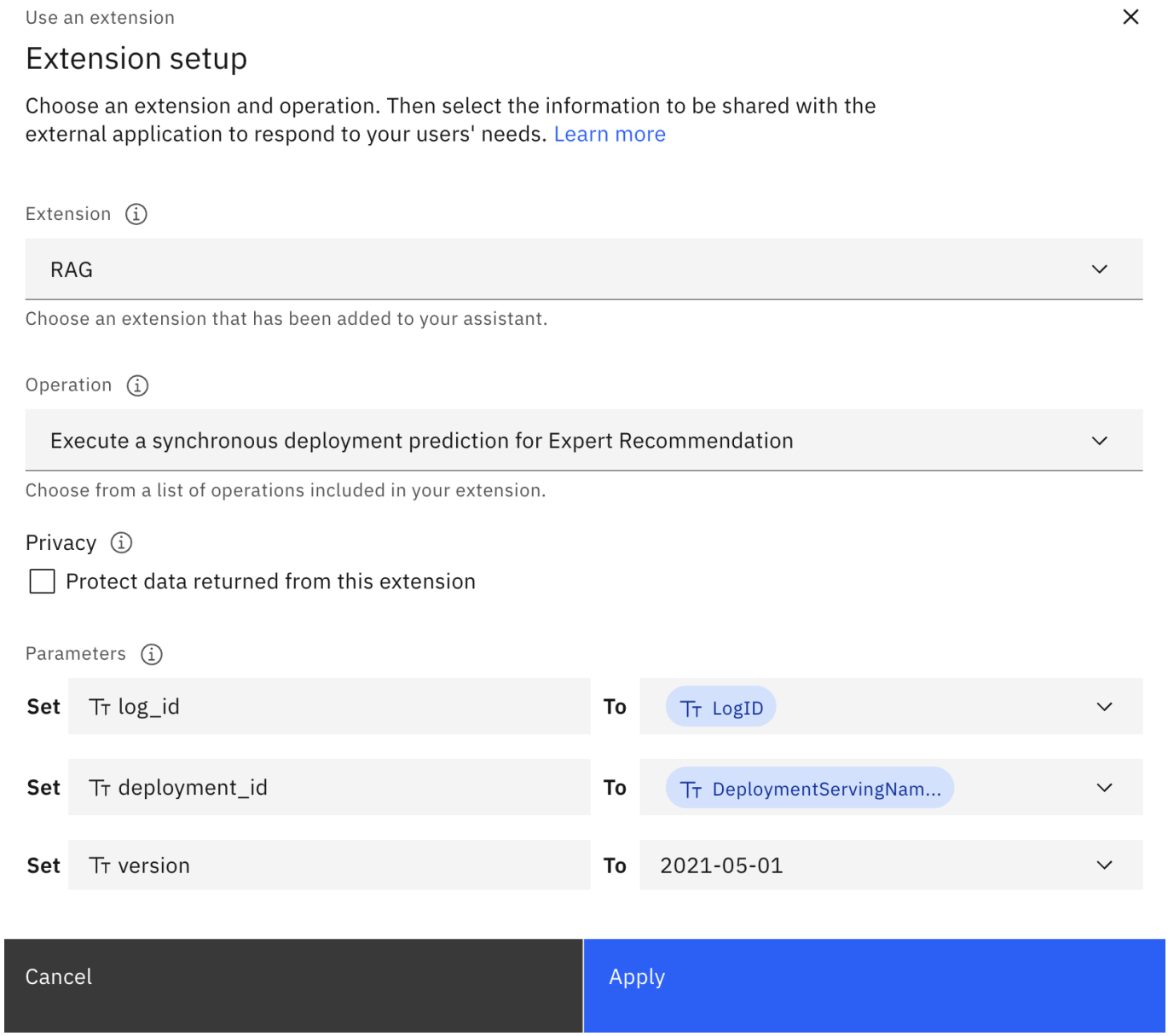


Figure: Extension setup for retrieving expert profiles

3. In the field Assistant says you can provide a message such as *"Fetching expert information based on your question."*
4. Under the ***Conversation Steps*** click ***New Step***.

5. Under this step click ***Set new value*** → ***Session variables*** → ***ExpertResult***, click ***Expression***. Enter the following expression:

```
{step_<1>_result_1.body.recommended_top_experts}[0]
```

where `${step_<1>_result_1.body.recommended_top_experts}` is the RAG extension response variable. Click **Apply** then **Save**. This step must be done similarly to the steps under the section **Add Step 3** section above under **Return the RAG extension response**.

6. Similarly set 5 more variable values, `ExpertID`, `ExpertName`, `ExpertEmail`, `ExpertRole` and `Domain` (This depends on what/how many fields you have ingested to vector database). However, in the ***Expression*** section for the `ExpertID` variable, enter the following expression and ***Save***:

```
${ExpertResult}.expert_id
```

where `${ExpertResult}` is the Session variable created in the previous step. After doing so click ***Apply***. Do the same for `ExpertName`, `ExpertEmail` `ExpertRole` & `Domain`. Add the ***Expression*** fields respectively as

```
${ExpertResult}.name
${ExpertResult}.email
${ExpertResult}.position
${ExpertResult}.domain
```

7. After setting up all the session variables & their ***Expression*** fields ***Save***. In the ***Assistant says*** section put all the new session variables created. Click ***Save*** (see image below for reference)

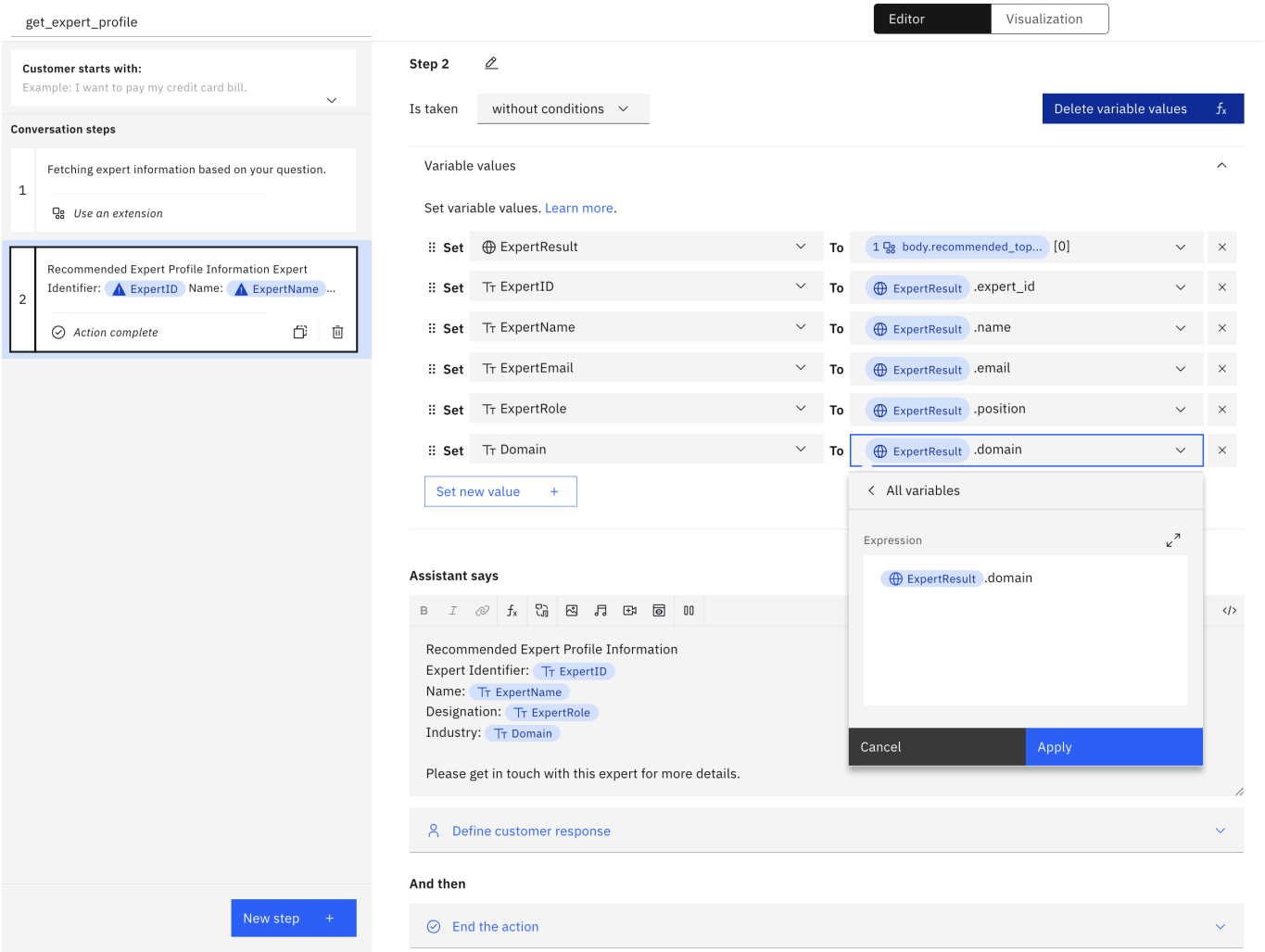


Figure: Assign and show expert profile information

Create action that asks for expert profile information

1. Navigate to ***Actions*** → ***All items*** → ***Created by you***, and click ***New action***, and the ***Custom-built action***. Do not enter an example (click Cancel). Enter action name `get_expert_details` on the Untitled Action field (input field at the left upper corner), and click on conversation steps 1.
2. In section ***Assistant says*** switch to JSON editor by clicking the `(</>)` button and enter the JSON content below. Click ***Save***.

```
{
  "generic": [
    {
      "label": "Get Expert Details",
      "value": {
        "input": {
          "text": "Get Expert Details"
        }
      },
      "button_type": "post_back",
      "response_type": "button"
    }
  ]
}
```

Call the Get Expert Profile action

Add this step to integrate the expert retrieval with the initial `help` Action.

1. Go to the toolbar, Navigate to ***Actions*** → ***All items*** → ***Created by you***. Open action **help** and click on conversation steps 2.
2. Click ***New step*** which ensures to add a step in between existing Conversation steps 2 and 3. As a result there will now be a total of 4 steps.
3. Under the newly created **Step 3** in section ***And then*** select ***Go to a subaction***. Select `get_expert_details` and click ***Apply***.
4. Finally ***Save***.

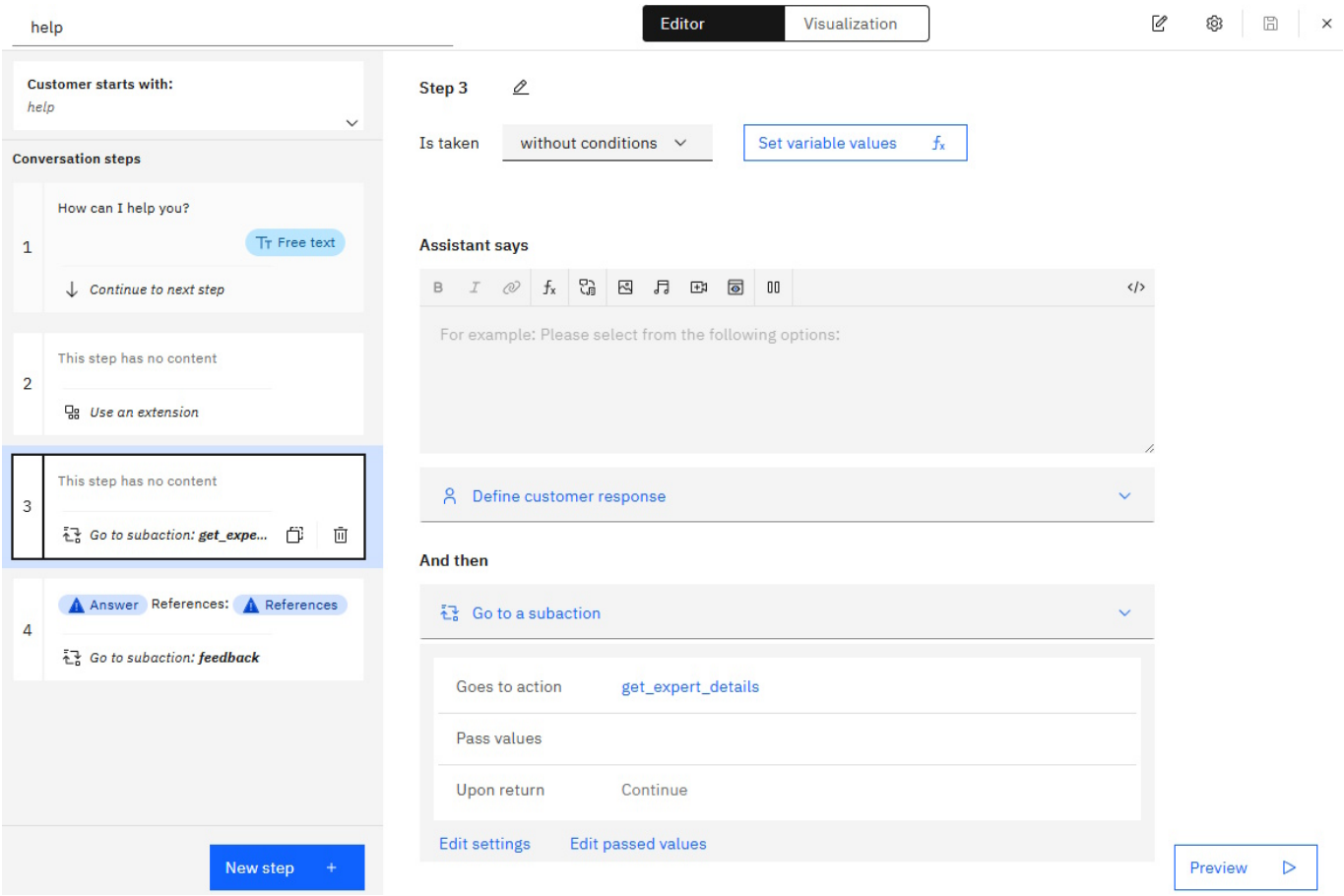
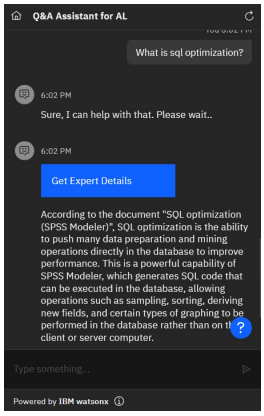
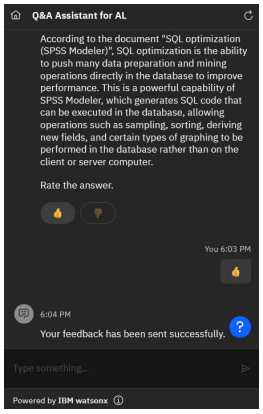
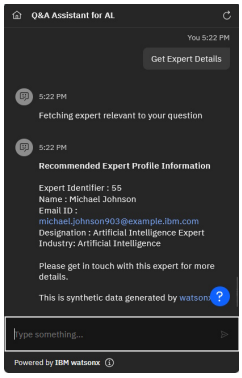
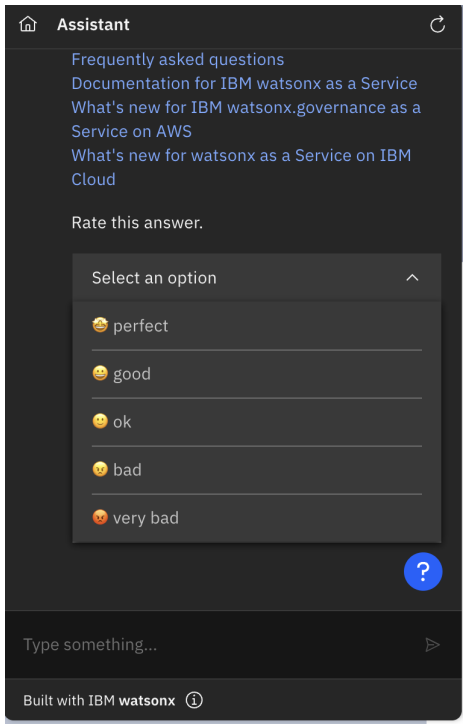


Figure: Add expert profile information to `help` action

Test RAG QnA with feedback and retrieval of expert profile information

Test the conversation flow on the ***Preview*** tab

Q&A using RAG	Feedback Logging	Expert Profile Retrieval	5 Star Feedback Logging

Q&A using RAG	Feedback Logging	Expert Profile Retrieval	5 Star Feedback Logging
			
Response from LLM	Postive/Negative	Get Expert Details button	5 Star Rating

Note : The below section is only applicable if you are using Elastic search and not Milvus as your vector database connection.

Check log record in Elasticsearch

Logon to Kibana and navigate to the log index in Elasticsearch. Find the document for the log record that corresponds to the conversation above. It contains, among others, the question and response, documents found and the user's feedback.

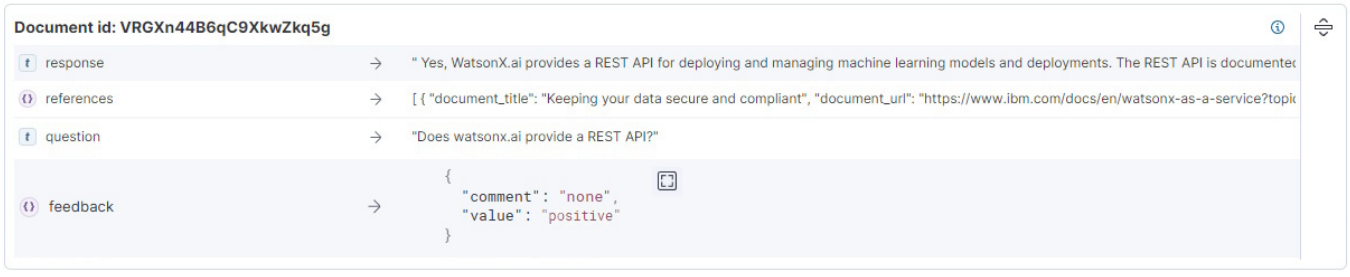


Figure: Elasticsearch document

License