

Steps to recreate the scenario:

Prerequisites:

1. **Set up a bot on Azure AI Studio:** For this specific example, the [Chat Playground](#) has been used to create and test a bot. You can read more about it [here](#). (Note: Store the URL and the API key for the bot as you will use it later to make HTTP calls to it.)
2. **Set up Microsoft Power Platform:** Create a M365 Developer account. You can read more about it [here](#).
3. **Set up SAP connection in Power Automate:** To connect Power Automate flows to your SAP system and make use of the SAP ERP connector for seamless data retrieval and updates, you must establish a connection between the Power Platform and your SAP system, below are the resources on how to do that.
 - [Power Platform + SAP - Installing the On-premises Data Gateway](#)
 - [Power Platform + SAP - Calling "Remote Function Call" \(RFC\) in SAP \(youtube.com\)](#)
4. **Set up / Sign up for Copilot studio.** Learn how to get started with it [here](#).

Now that all the prerequisites are in place, we can now begin to build the scenario.

Let us begin by getting started with the Copilot Studio:

The first step is to create a new Copilot, to do this you can go to the settings as shown below.

The screenshot shows the SAP Order Status Copilot settings page in Copilot Studio. The top navigation bar includes 'Overview', 'Knowledge', 'Topics', 'Actions', 'Analytics', 'Channels', 'Published 5/21/2024', 'Publish', 'Settings', and 'Test'. A central callout box says 'Your copilot is ready! Here's what's next:' with three items: 'Add actions so your copilot can do things for you', 'Build topics to focus and guide how your copilot answers', and 'Publish your copilot so others can use it'. Below this is a 'Details' section with a 'Name' field containing 'SAP Order Status Copilot' and a 'Description' field with 'None provided'. On the right, there is a 'Test your copilot' button and other interface elements.

Go to the Generative AI section as shown below and enable “Generative AI” (this is what automatically detects which topic/flow to call based on the user prompt):

S

The screenshot shows the Microsoft Copilot Settings page. On the left, there's a sidebar with various options: Copilot details, AI integration tools, **Generative AI** (which is selected and highlighted in blue), Security, Entities, Skills, Languages, and Language understanding. The main content area has a "Save" button at the top. Below it, the "Generative AI" section is titled "Generative AI". It contains a brief description of the feature, links to pricing tiers and preview terms, and a link to learn more about responsible AI. A sub-section titled "Use AI features in your copilot" notes that generating responses using AI doesn't guarantee accuracy or relevance. It includes two sections: "How should your copilot decide how to respond?" (with a "Classic" mode radio button and a "Generative (preview)" mode radio button, the latter being selected) and "Intelligent authoring with Copilot" (with a link to go to user settings). At the bottom, there's a "Copilot content moderation" section with a dropdown set to "High (default)" and a note about generating fewer answers.

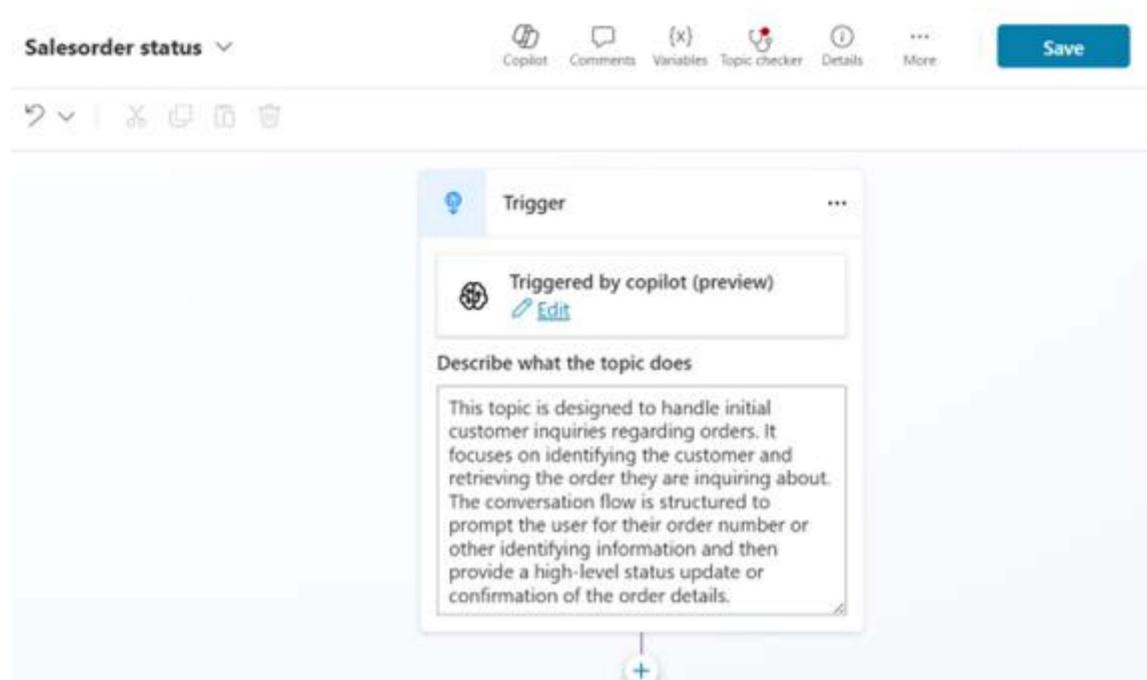
The next step is to create a new topic and the name is – “Salesorder status” – this is the topic that the salesperson will use to get the order status and information on the order.

The Copilot studio has unique functionality which figures out which topic/functionality to call based on the input text sent to the copilot/bot. This functionality is called “Dynamic Chaining” and uses AI to automatically pick the topic that fits the best based on the description of the topic provided while creating it. This is essentially the equivalent of automatically knowing which function in the code to call based on a sentence!

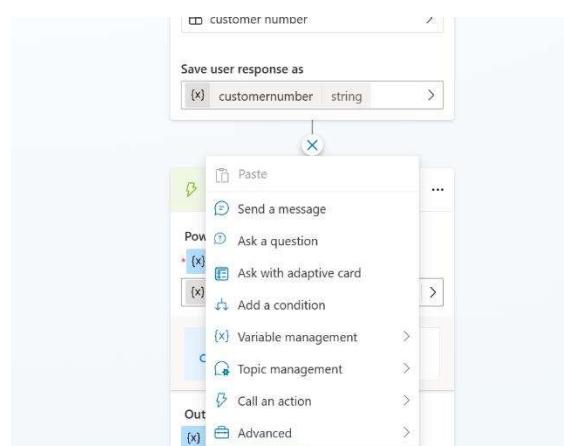
Therefore, the first step after creating a new topic is to write a description of the topic. This is what will help the AI identify what this flow is for.

Here is a link to the guidelines to create the description for the topic that will help the AI best understand the intent behind what you want to build: [Use topics to design a copilot conversation - Microsoft Copilot Studio | Microsoft Learn](#)

Here is the image of the topic description used for this. (Fun fact- Copilot was used to write the description)



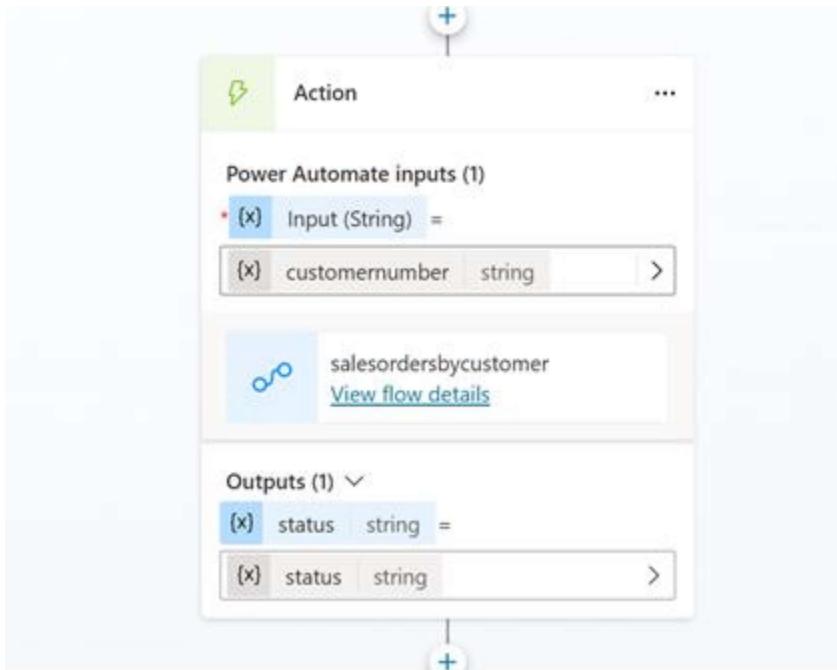
The next step is to add a "question node" to ask the user what customer number they want this information for.



After the user responds, we want to identify the customer number from the user response, therefore in the question node, we will create an entity with the regex (regular expression) matching a customer number (this can be changed to whatever the most generalized pattern is for your system).[Here](#) is a link on learning more about creating entities.

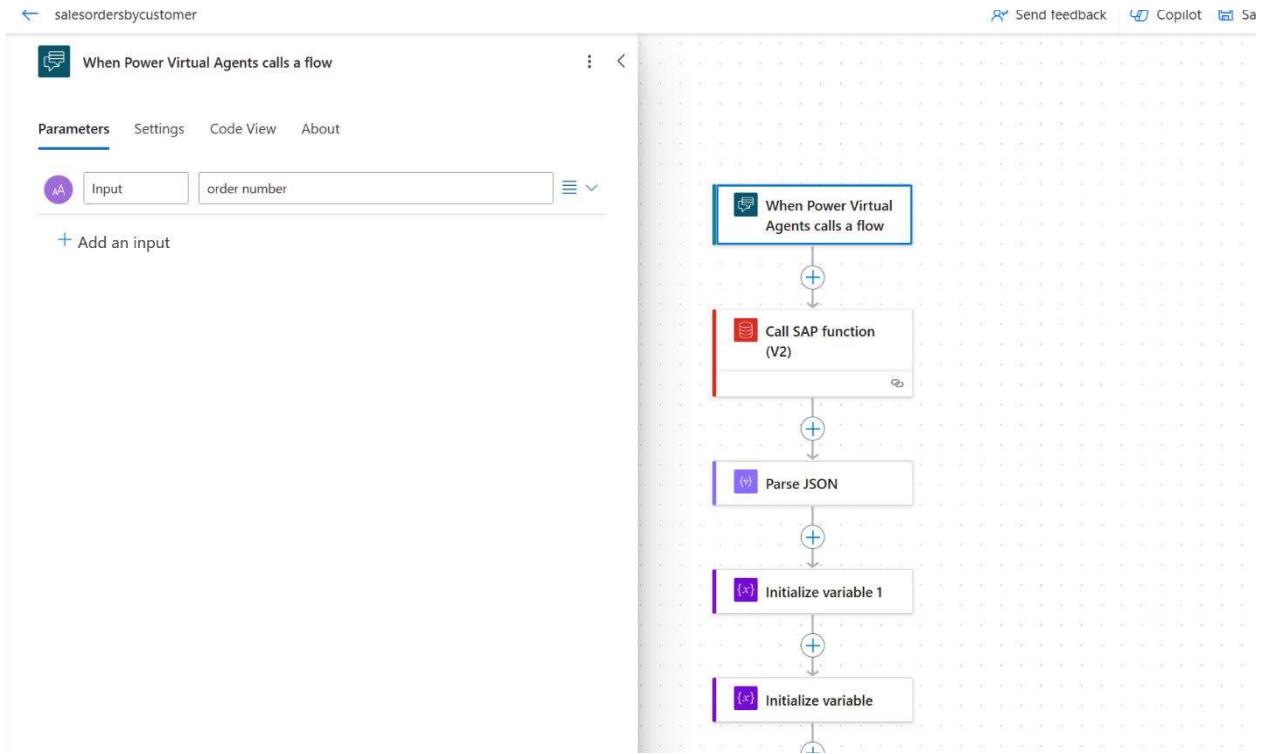
The screenshot shows the Microsoft Power Automate interface with a 'Trigger' card at the top. Below it is a 'Dynamic chaining (preview)' card with a 'Edit' button. A 'Question' card is selected, containing the text: 'What is the customer number you wa...'. Under 'Identify', there is a dropdown menu set to 'customer number'. In the 'Save user response as' section, 'Var1' is selected as a string variable. An 'Action' card is at the bottom. A modal window titled 'Choose information to identify' is open on the right, asking to help the copilot pick out specific information from the user's response. It includes a search bar and a list of options: 'Create an Entity', 'Multiple choice options', 'Prebuilt String', 'Options from a list variable', 'Table', 'customer number', 'sales order number', and 'User's entire response'.

The next step is to now call an action and create a Power Automate flow that gets and parses this information from the SAP system.



Below is the description of the first Power Automate flow to get all the orders for each customer with each step explained. This flow is called "salesorderbycustomer".

The input to the Power Automate flow will be the customer number extracted from the input from the user.



The next step will be to call the SAP BAPI (BAPI_SALESORDER_GETSTATUS). For this, you must provide the customer number as well as the sales organization number. You would also have to enter your SAP server credentials (as shown in the SAP set up videos linked above in the prerequisites.).

[← salesordersbycustomer + Published](#)

[Send feedback](#) [Copilot](#)

Call SAP function (V2)

Parameters Settings Code View Testing About

SAP System *

RFC Name *

BAPI_SALESORDER_GETLIST

Advanced parameters

Showing 2 of 3

RFC Group Filter

*

Auto Commit

No

```
graph TD; A[When Copilot Studio calls a flow] --> B[Call SAP function (V2)]; B --> C[Parse JSON]
```

Once you call the SAP BAPI, you will get a JSON response. This response needs to be parsed and stored so that we can use the information to perform further tasks. To do this, you must provide a sample JSON. The JSON is available on GitHub with the exported Power Automate flow.

The screenshot shows the 'Parse JSON' tool interface. At the top, there are tabs: 'Parameters' (which is selected), 'Settings', 'Code View', and 'About'. Below these are two sections: 'Content *' and 'Schema *'. The 'Content *' section contains a preview of the JSON data with the file name 'SALES_O...'. The 'Schema *' section displays the JSON schema:

```
{
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "SP_DOC": {
          "type": "string"
        },
        "ITEM_NUMBER": {
          "type": "string"
        },
        "MATERIAL": {
          "type": "string"
        },
        "SHORT_TEXT": {
          "type": "string"
        },
        "DOC_TYPE": {
          "type": "string"
        },
        "DOC_DATE": {
          "type": "string"
        }
      }
    }
  ]
}
```

To the right of the tool interface is a flowchart diagram. It starts with a trigger 'When Copilot Studio calls a flow', followed by a 'Call SAP function (V2)' step, then a 'Parse JSON' step (which is highlighted in blue), and finally an 'Initialize variable 1' step.

The next step is to initialize two variables – one called “allorders” which has the type “array”, and its values will be the body of the JSON we retrieved previously.

The screenshot shows the 'Initialize variable 1' step in a Power Automate flow. The step is labeled '{x} Initialize variable 1'. Below it is a configuration panel with tabs: 'Parameters' (selected), 'Settings', 'Code View', and 'About'.

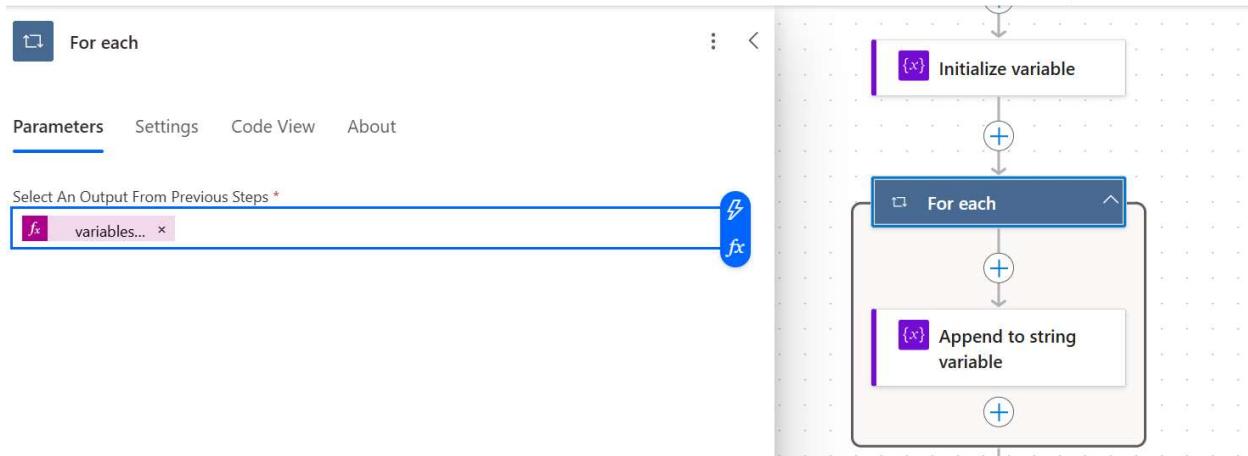
The 'Parameters' tab contains the following fields:

- Name ***: allorders
- Type ***: Array
- Value**: {x} Body x

The other variable is the string variable we will use to store all the parsed fields/information from the sales documents we need to show the user called and is called "orderdetails."

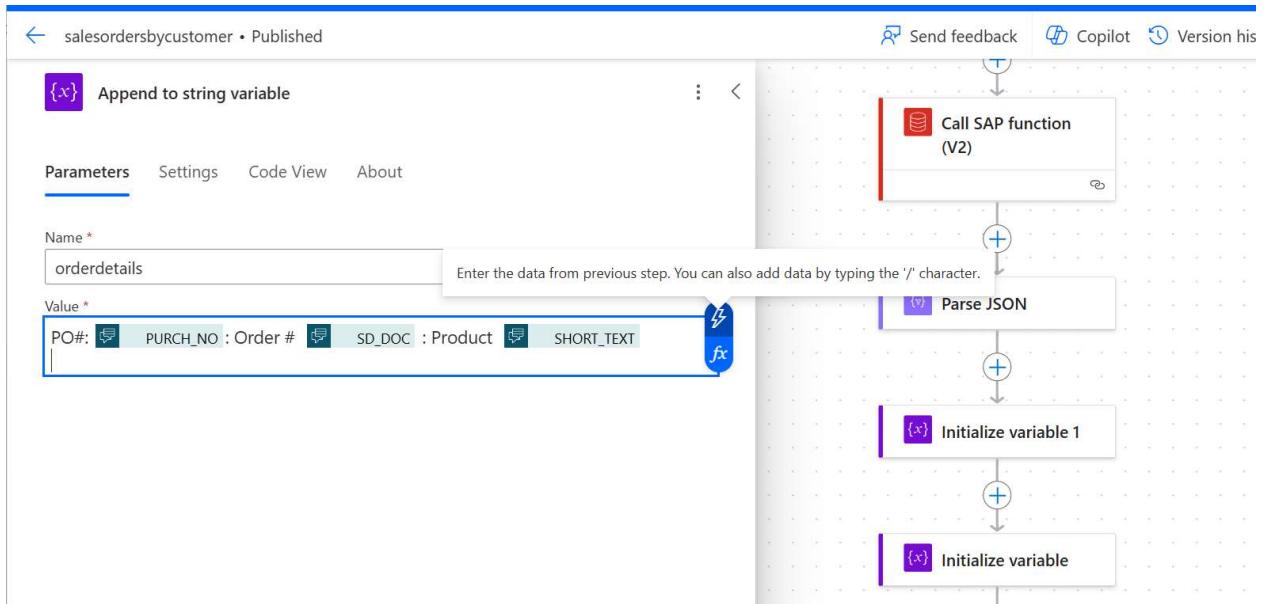
The screenshot shows the 'Power Automate' interface with a blue header bar. Below it, a navigation bar includes a back arrow, the text 'salesordersbycustomer • Published', and a search bar. The main area is titled '{x} Initialize variable'. A toolbar with three icons is visible on the right. Below the title, there are tabs: 'Parameters' (which is selected), 'Settings', 'Code View', and 'About'. The 'Parameters' section contains three input fields: 'Name *' with the value 'orderdetails', 'Type *' set to 'String', and 'Value' with the placeholder 'Enter initial value'.

Next, we need to loop over all the sales documents of the customer, go over each and then take all the fields we need from it, and finally paste it in the response. For this, we will use the for each loop and go over the variable "allorders".



Inside the foreach loop, we will append the fields as shown below to the variable "orderdetails".

You can add these dynamic fields from the dynamic variables shown below.



Once the loop has completed and all the information regarding the documents has been appended to the string variable "orderdetails", we will send that variable back to the Copilot Studio with all the orders for that customer.

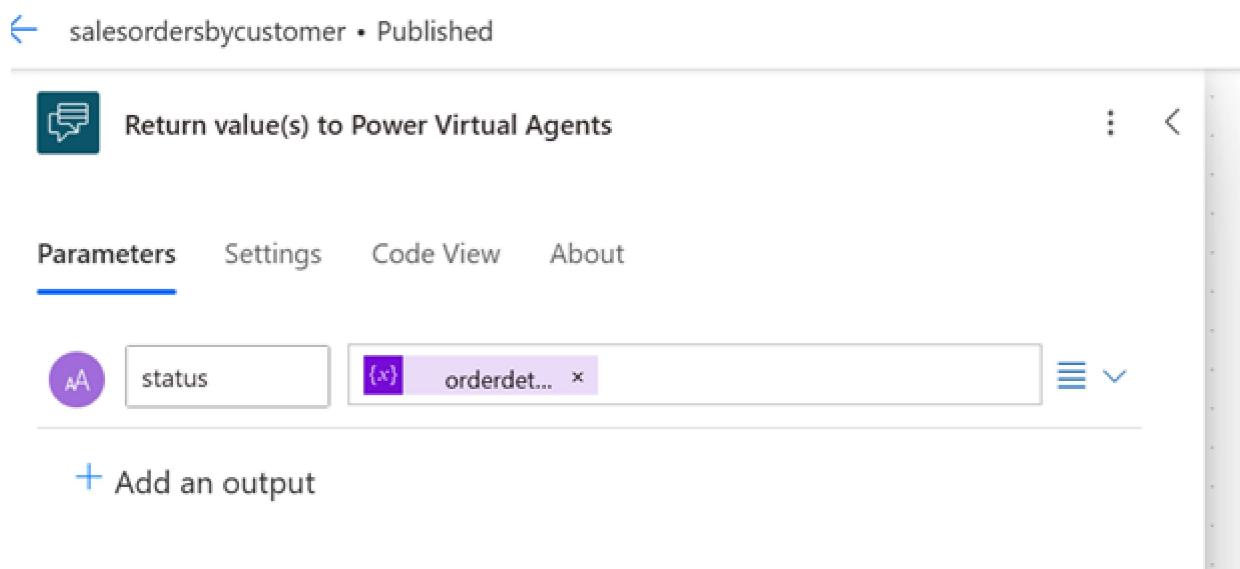
← salesordersbycustomer • Published

Return value(s) to Power Virtual Agents

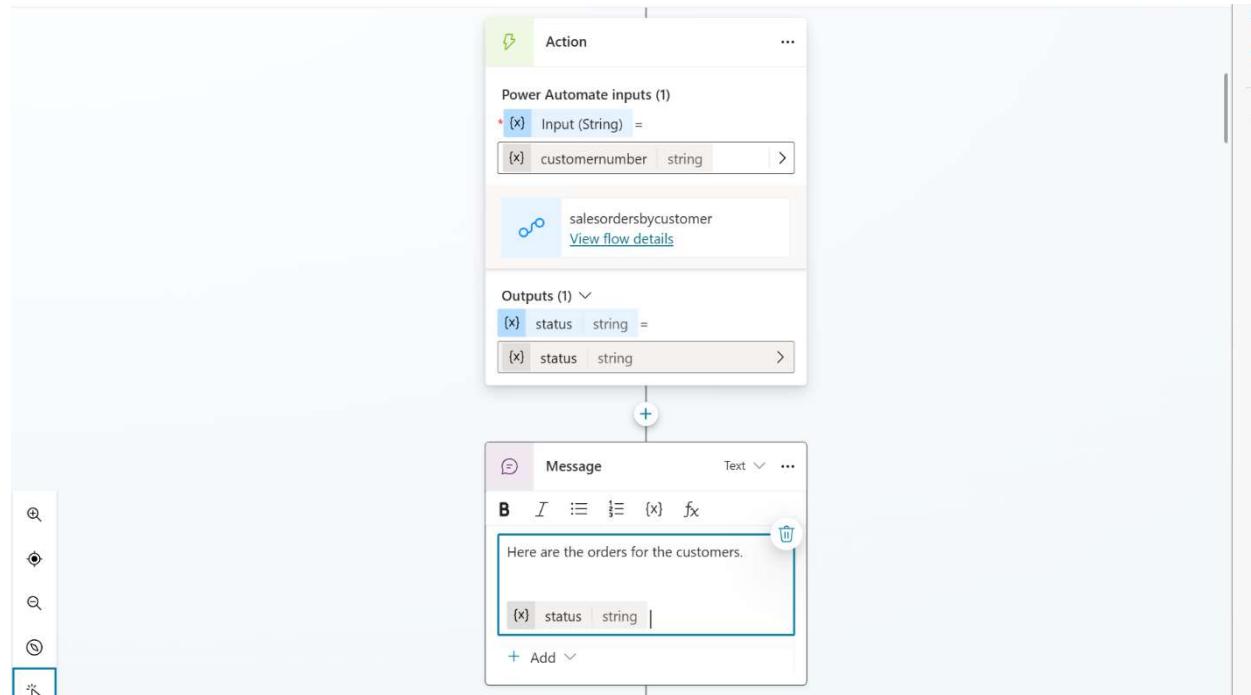
Parameters Settings Code View About

status {x} orderdet... ×

+ Add an output



Now back in the Copilot Studio we will append the response returned by the Power Automate flow as shown like this.

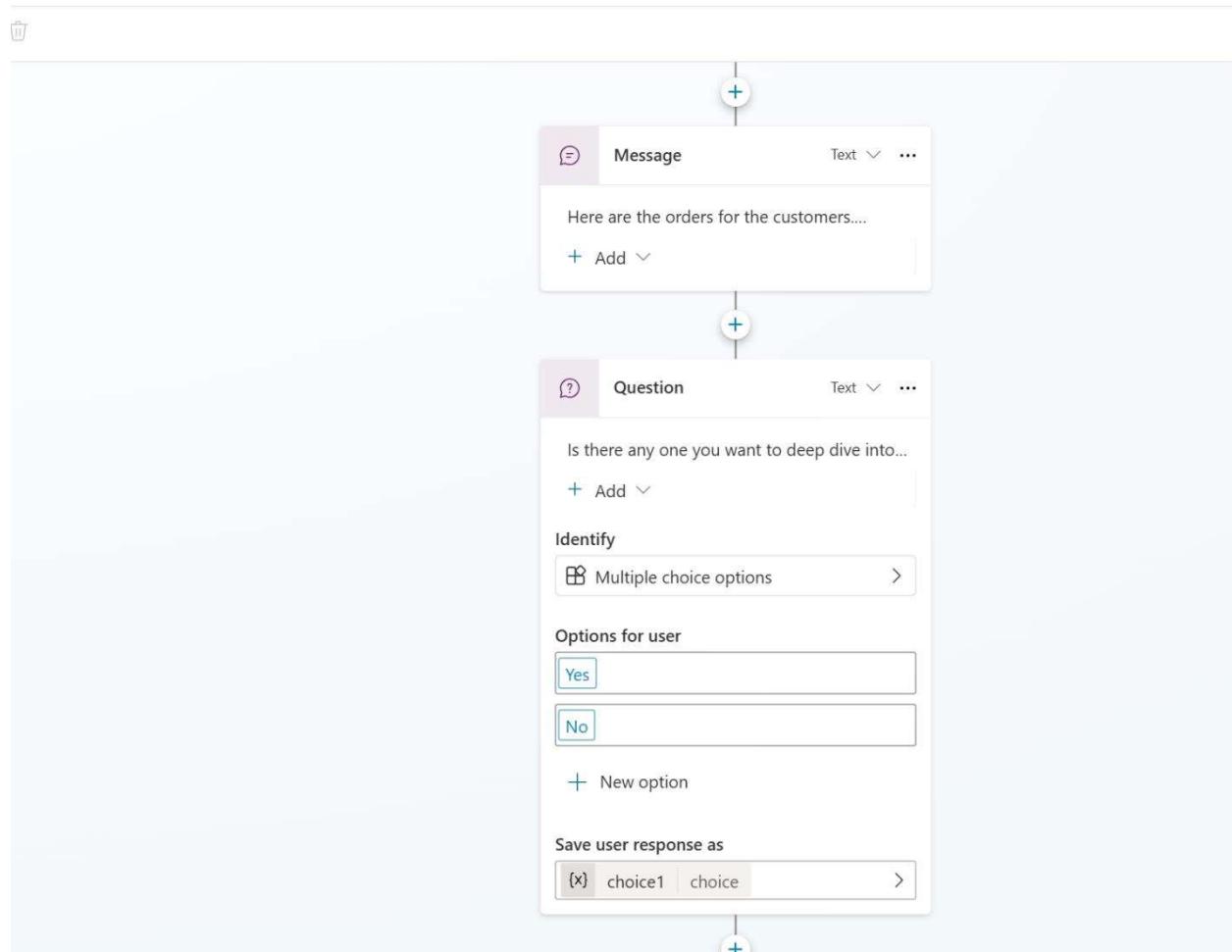


The screenshot shows the Copilot Studio interface with the following components:

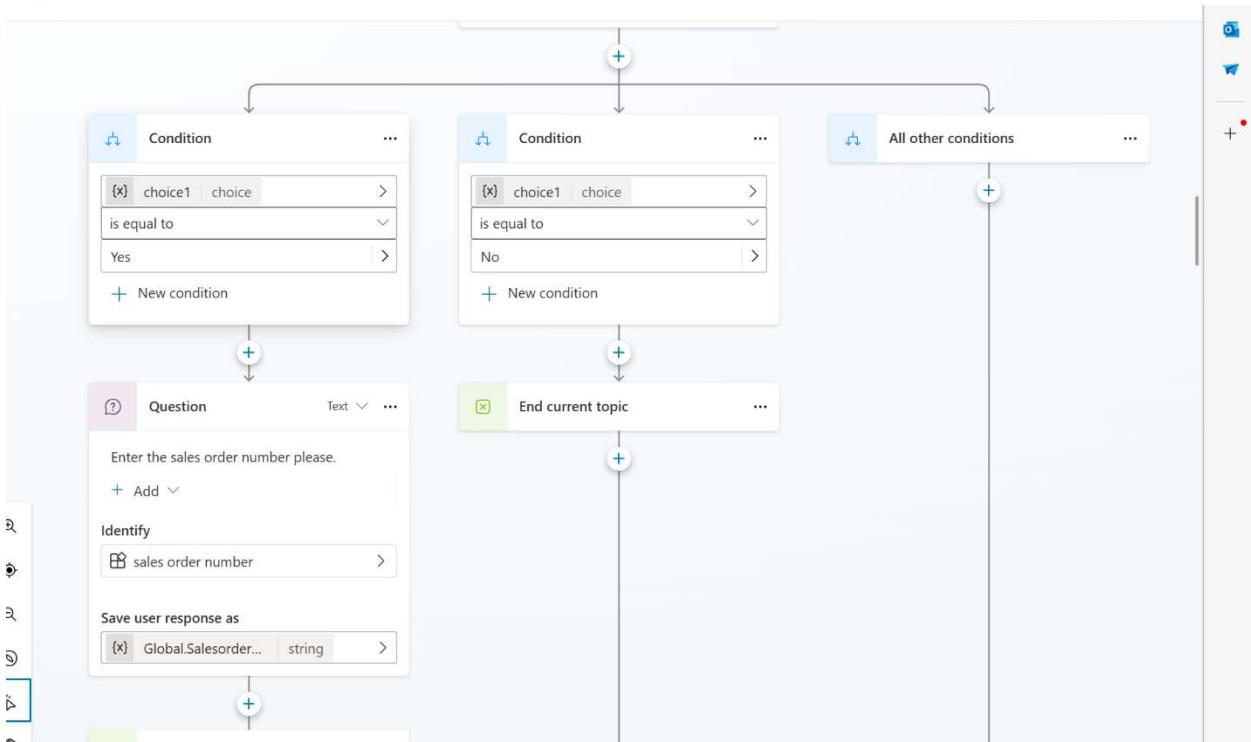
- Action:** A Power Automate action block with the following settings:
 - Power Automate inputs (1):** An input named "customernumber" of type string.
 - Outputs (1):** An output named "status" of type string.
- Message:** A message block containing the text "Here are the orders for the customers." followed by a placeholder for the "status" variable.

Once we have shown the salesperson all the customers' sales orders, they can now act on it and deep dive into the sales order they want to look into.

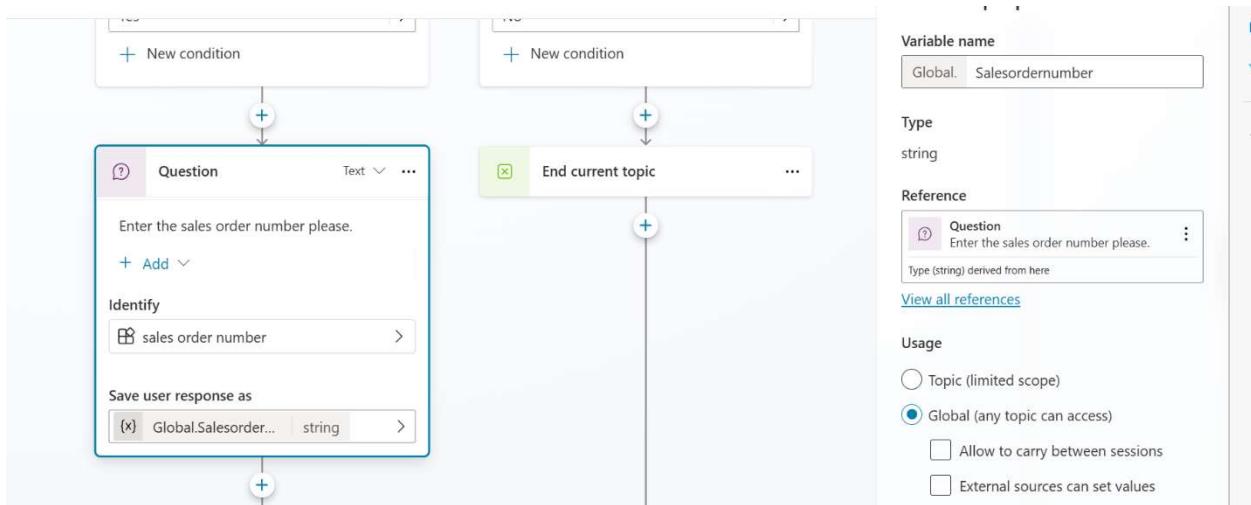
Note: In this demonstration one such action that the salesperson could take has been shown. The action we have selected for this demo is asking the salesperson if there is any particular problematic order they want to dive into and confirm the status of. However, you could create any such flow based on what you would like the functionality to be.



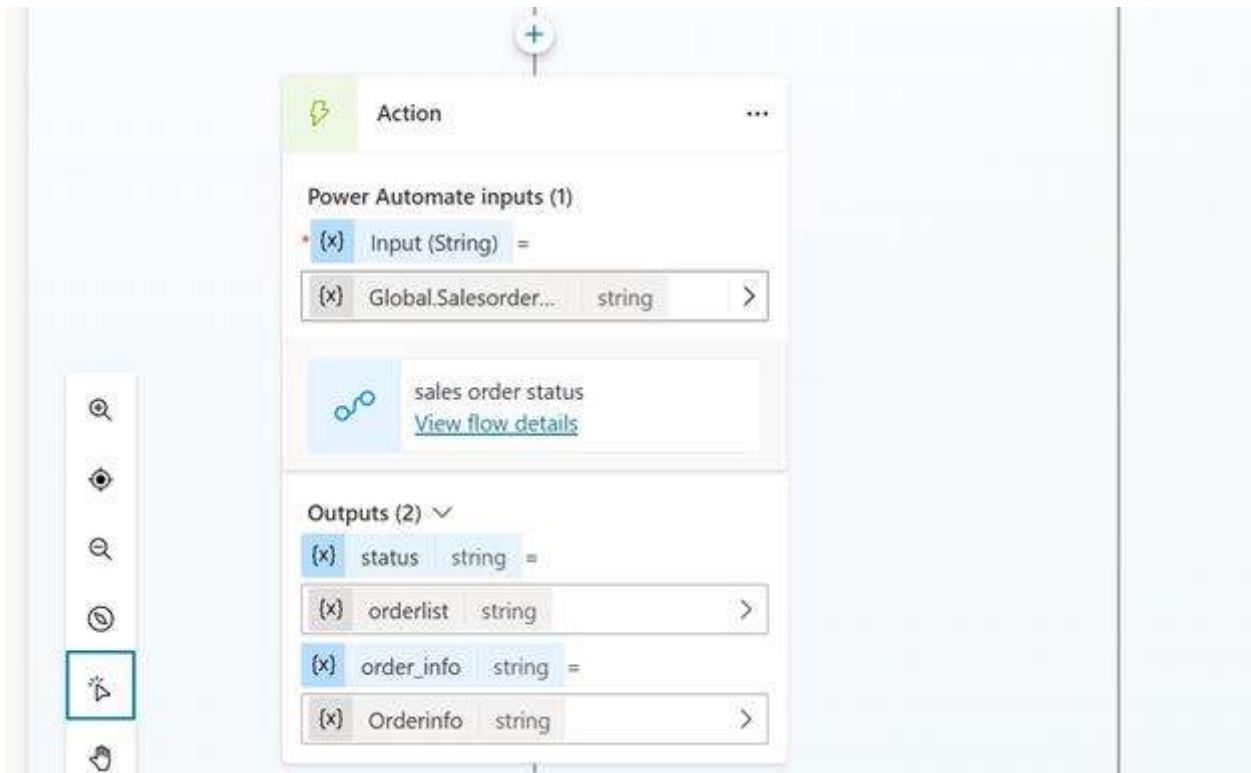
We will create a question node and ask the salesperson to enter the order number in the Copilot studio. We will once again create a new entity with the regex in the format of the SAP sales order number (the same way we did it for the customer number above).



The sales order number the customer wants to know about will be a global variable as this may be valuable information across topics. You can make it a global variable by creating a new variable and modifying it in its settings.



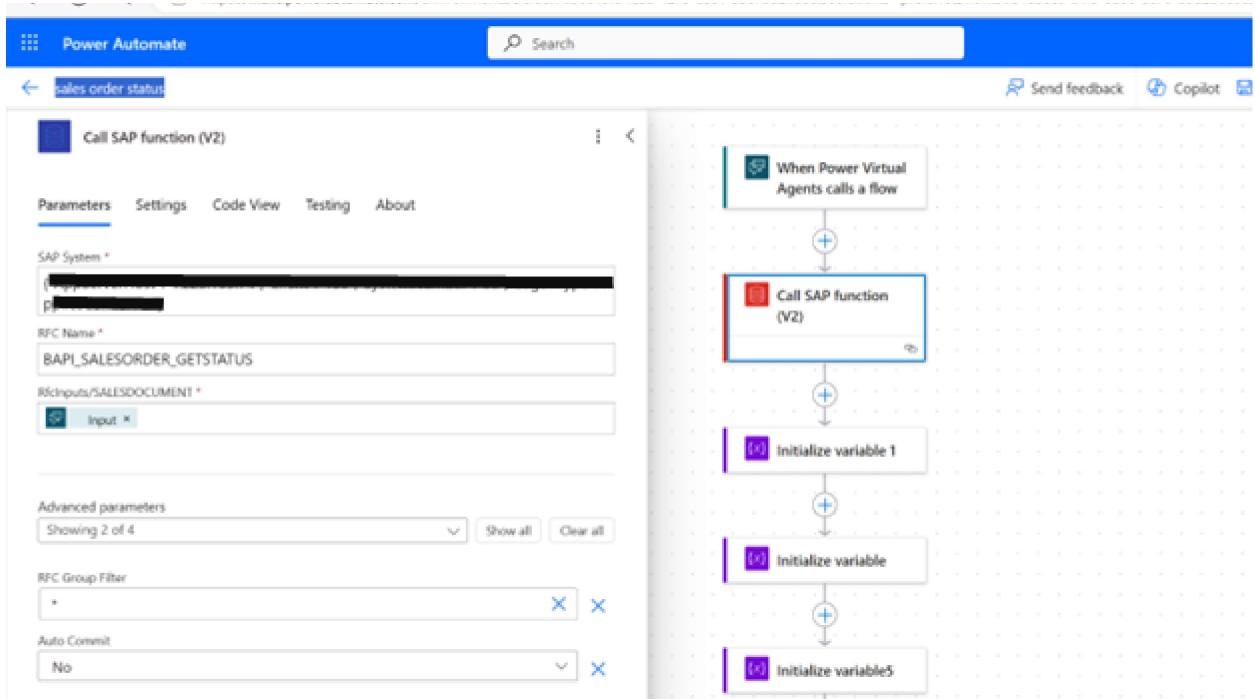
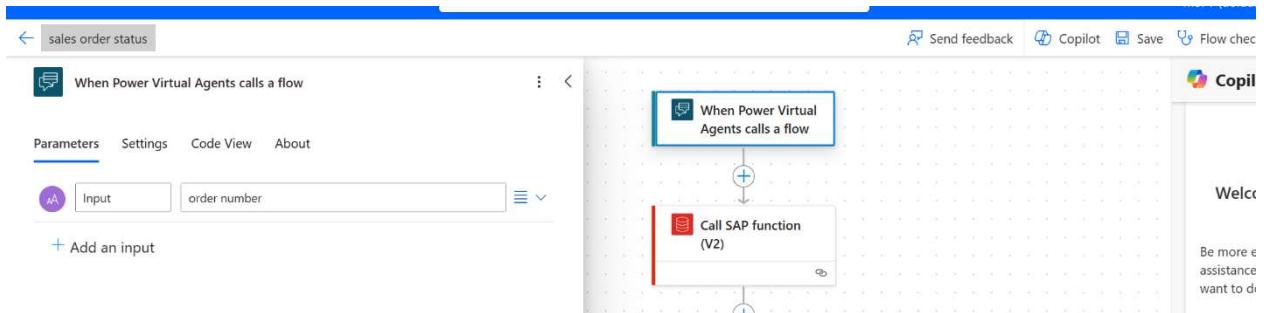
Once we have the sales order number that the customer wants to use, we will now dive deep into this sales order and find out about its status from the SAP system and find out whether it is delayed or not.



Therefore, the next Power Automate flow we need to call is one that gets the order status for the order the customer mentioned and returns a response on whether the order is delayed or on time. The input to the flow will be the sales order number.

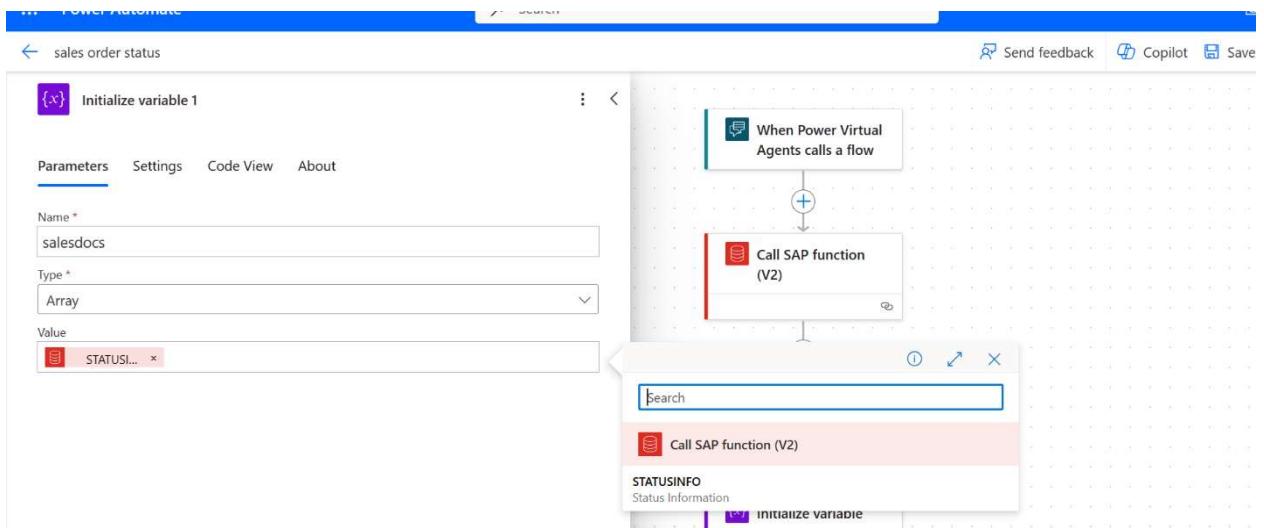
Here are the steps for this Power Automate flow "sales order status."

We will first take the order number as an input to the Power Automate flow, next, we call the BAPI_SALESORDER_GETSTATUS – this is to get the order status information for the sales order.



Next, we initialize three variables.

"salesdocs" – of type array and stores the sales document response from the SAP system.



[!\[\]\(3488592b4a8f4d31e7880a75336bfbc7_img.jpg\) sales order status](#)



Initialize variable



Parameters

Settings

Code View

About

Name *

deliverystatus

Type *

String



Value

Enter initial value

"deliverystatus" of type string which will store the delivery status of the order.

The screenshot shows a software interface for managing variables. At the top, there is a blue header bar with a back arrow and the text "sales order status". Below this is a toolbar with a purple button containing the placeholder code `{x}`, followed by the text "Initialize variable2". To the right of the toolbar are three small icons: a vertical ellipsis, a left arrow, and a right arrow.

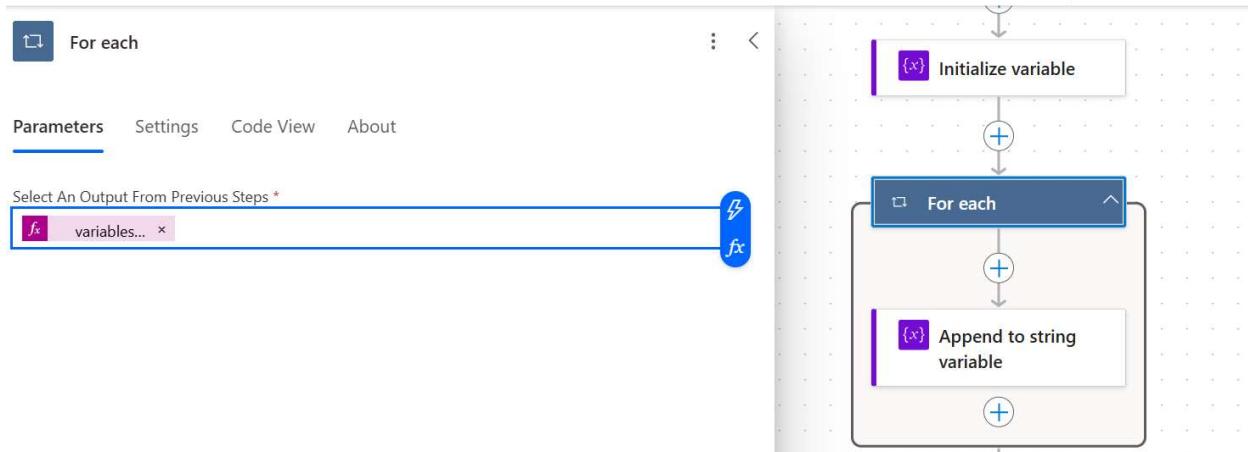
The main area has a tab-based navigation bar at the top with four tabs: "Parameters" (which is selected and underlined), "Settings", "Code View", and "About".

The "Parameters" section contains three input fields:

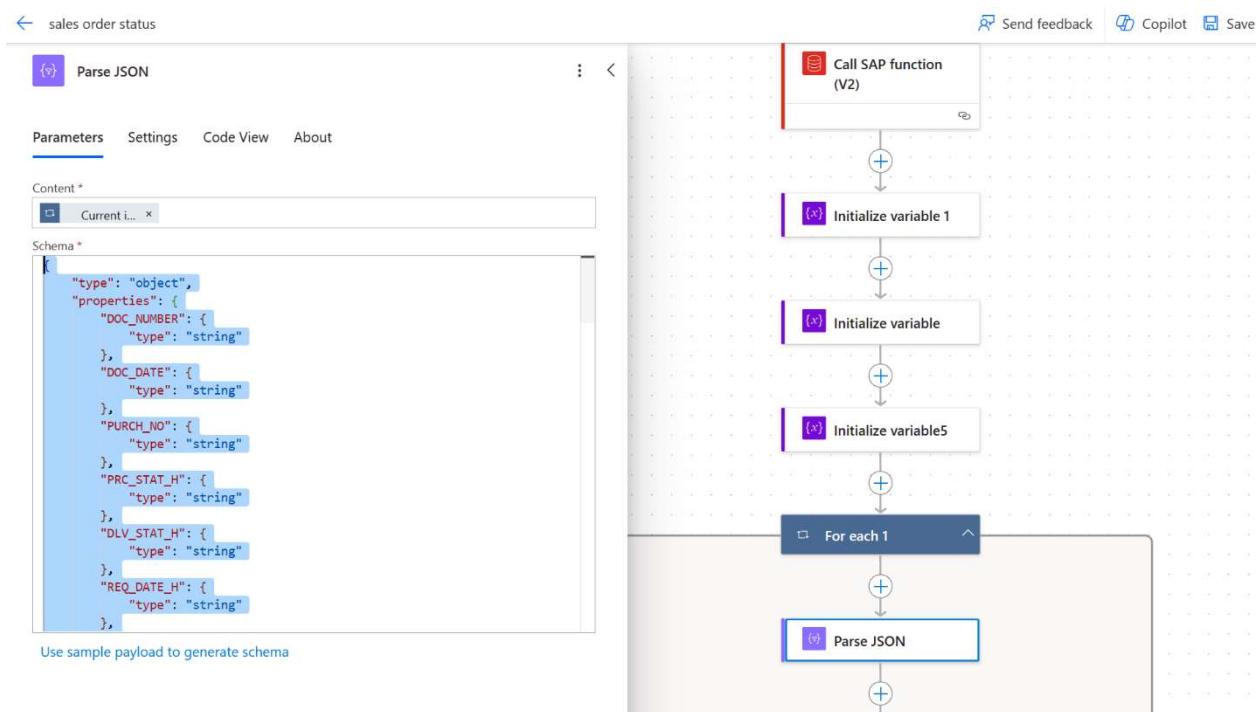
- Name ***: The input field contains the text "orderinformation".
- Type ***: The input field contains the text "String".
- Value**: The input field contains the placeholder text "Enter initial value".

And the variable "orderinformation" which will store other information about the order.

We will then create a for loop for each that would parse all the sales documents stored in the variable "salesdocs" (Sometimes an order has > 1 materials and therefore has two documents associated with the same order number). From the dynamic variables, pick the "salesdocs" variable to loop over.



Next, like the previous Power Automate flow, we must parse the JSON response.

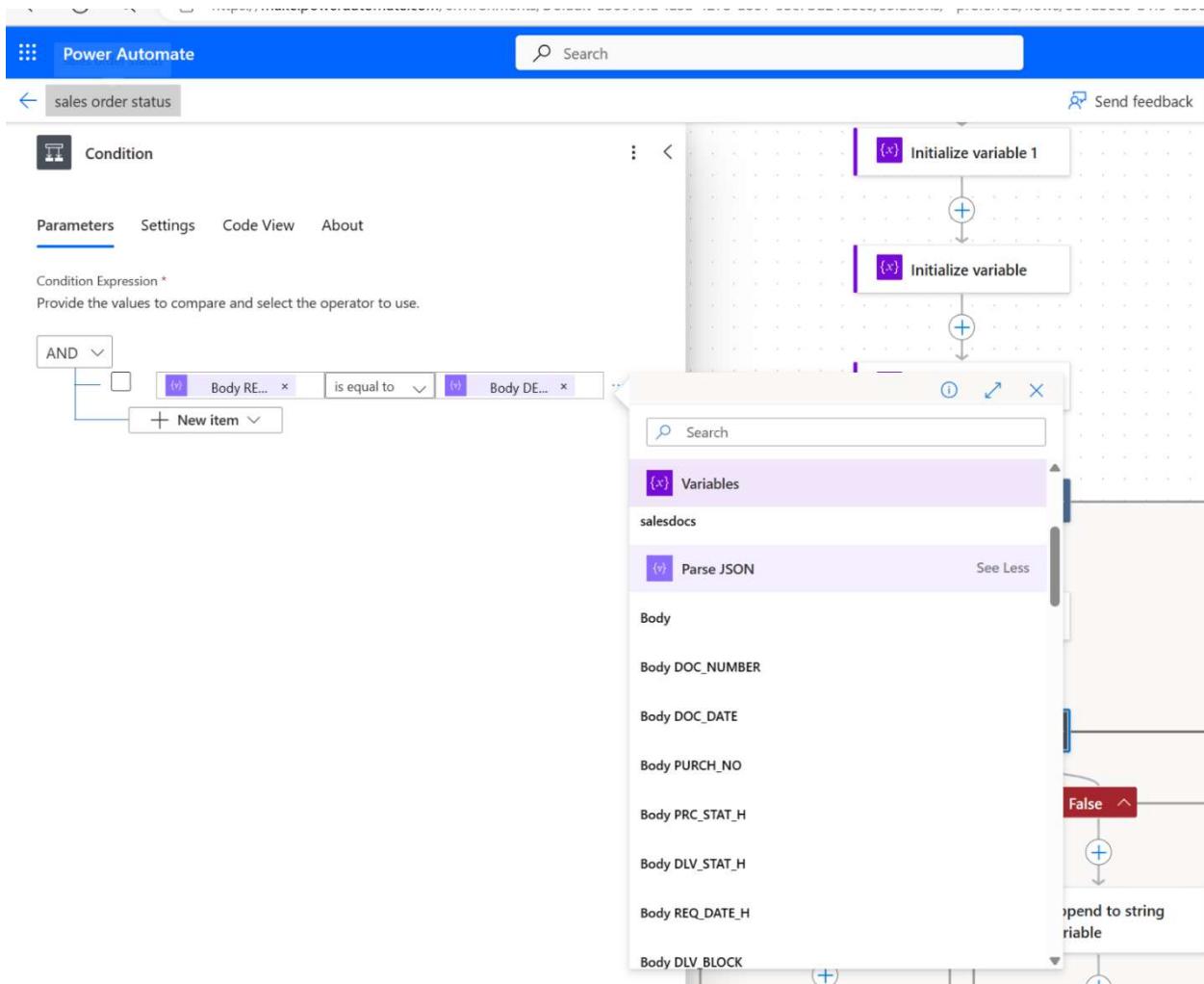


The JSON used for this is available in the GitHub with the exported Power Automate flow.

Next to check if the order is delayed, we will check if the delivery date of the order is the same as the requested date. If it is not, then the order is delayed.



To include this logic in the Power Automate flow, we will add a condition that the field Body.REQ_DATE is EQUAL to DELIV_DATE.



If this condition is true:

The text: "on time" would be appended to the variable "deliverystatus".

If the condition is not true and the order is delayed, the variable "deliverystatus" will be set to "delayed" the text as shown below will be appended to the variable "orderinformation".

{x} Set variable ⋮ <

Parameters Settings Code View About

Name *

deliverystatus

Value *

delayed

{x} Append to string variable ⋮ <

Parameters Settings Code View About

Name *

orderinformation

Value *

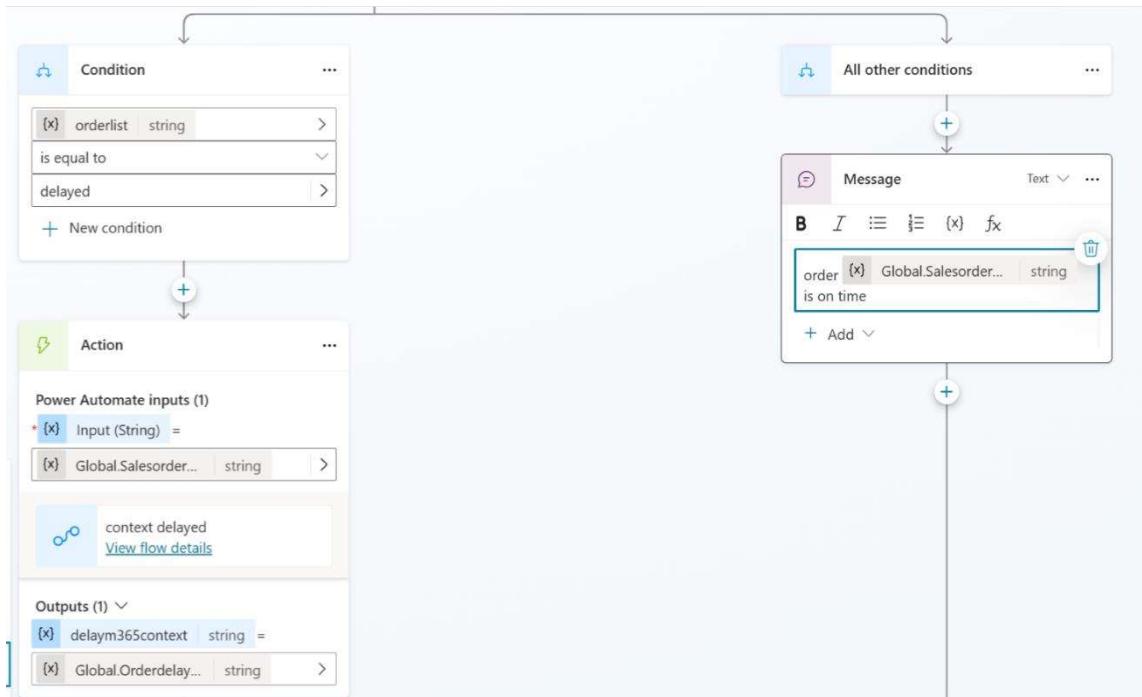
Customer order # {x} Body DO... × for {x} Body RE... × {x} Body SH... × with
a desired delivery date of {x} Body RE... × is still not delivered.

The values can be found in the flow once you download and export it from Github.

Both the variables "deliverystatus" and "orderinformation" will be returned to the Copilot Studio as shown in the Copilot Studio.

The screenshot shows the 'Return value(s) to Power Virtual Agents' configuration screen in the Copilot Studio. At the top, there is a title bar with a message icon and the text 'Return value(s) to Power Virtual Agents'. Below the title bar, there are tabs for 'Parameters', 'Settings', 'Code View', and 'About'. The 'Parameters' tab is selected. There are two parameter entries: one for 'status' with value '{x} deliverys...', and another for 'order info' with value '{x} orderinf...'. Both entries have a purple circular icon with 'AA' and a delete button. At the bottom left, there is a blue plus icon and the text '+ Add an output'.

Now in the Copilot Studio we will deal with both cases, whether the order is delayed or not by adding a condition node.



If the order is delayed, the next step is to harness the power of AI and the M365 data for conversation context (emails and Teams' chats on why the order is late). This is done by querying the M365 graph for Outlook emails and Teams' chats related to the order and then passing those emails and messages to Azure OpenAI. Azure OpenAI will then summarize the content and finally give the user direct insights, which saves the salesperson the time they would have taken to go over all the messages and emails manually.

Here is the description and steps for that Power Automate flow, the name of this flow is "context delayed":

The input for this flow will be the sales order number.

We will define two variables: "messages" and "emails"

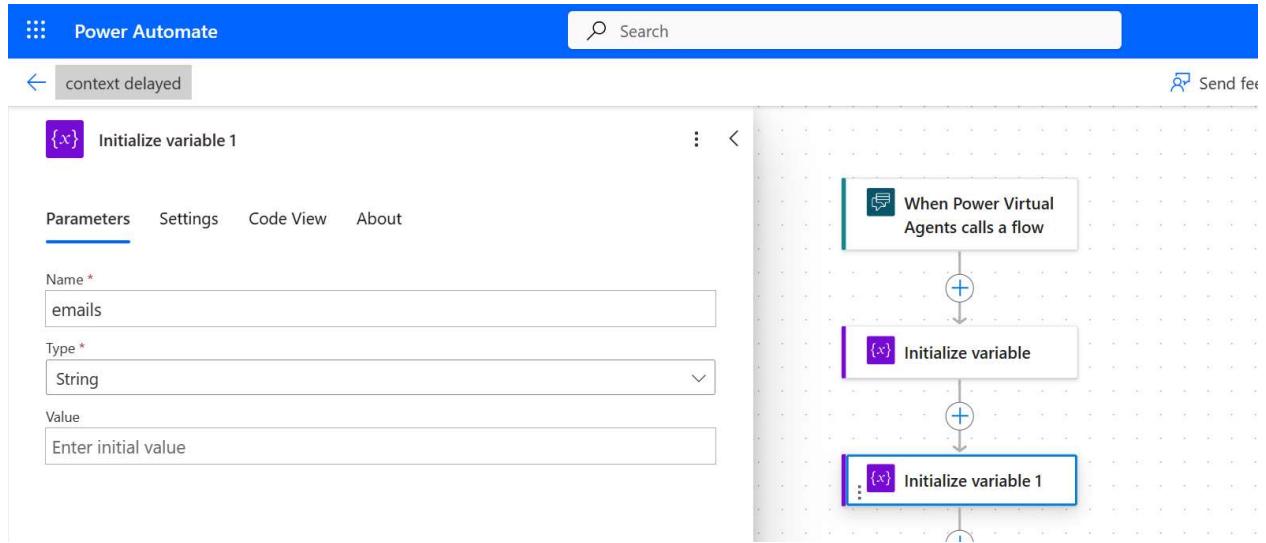
The screenshot shows the Power Automate interface. On the left, a configuration pane for an 'Initialize variable' step is open, showing the following details:

- Name ***: messages
- Type ***: String
- Value**: Enter initial value

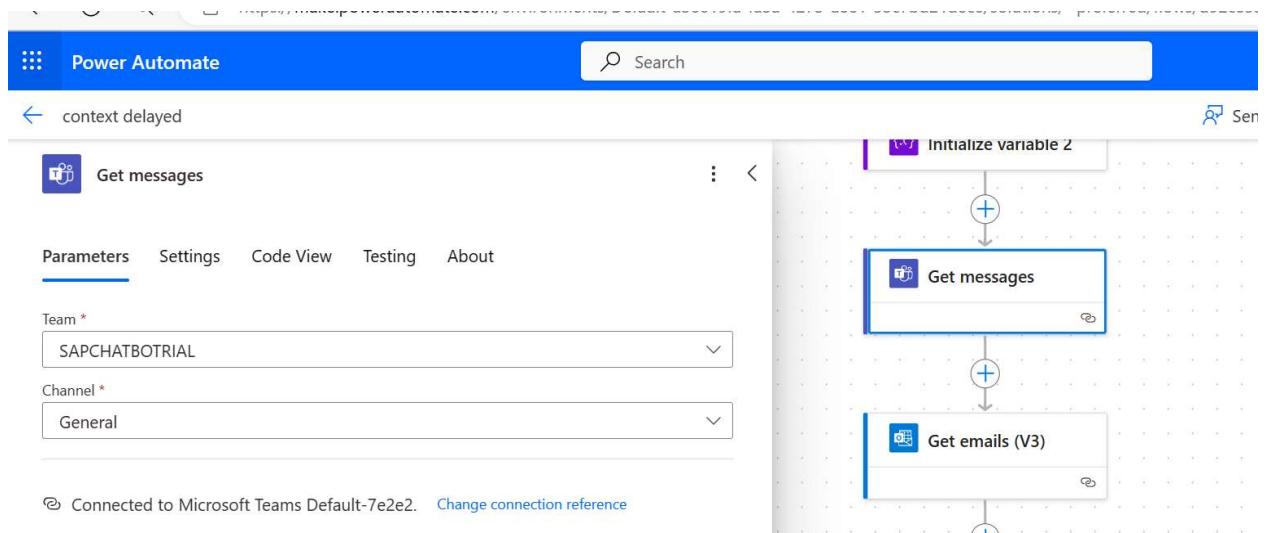
On the right, the main flow editor displays the following sequence of steps:

```
graph TD; Start[When Power Virtual Agents calls a flow] --> Init1[Initialize variable: messages]; Init1 --> Init2[Initialize variable 1]
```

The 'Initialize variable' step for 'messages' is highlighted with a blue border.



Next, we will also make use of the Team's connector as shown below, however the Teams connector does not yet allow you to query specific messages, therefore there will be a bit more processing on that front as shown below, for now we will just fetch all the messages.



Next, we will fetch all the outlook emails related to the order using the input order number as the search query .

The screenshot shows the Microsoft Power Automate interface. On the left, the 'Get emails (V3)' action is selected, showing its parameters. The parameters include:

- Importance: Any
- Only With Attachments: No
- Folder: Inbox
- Fetch Only Unread Messages: No
- Include Attachments: No
- Search Query: Input (with a placeholder icon)

On the right, the full flow is visible:

```
graph TD; Init[Initialize variable 2] --> GetMessages[Get messages]; GetMessages --> GetEmails[Get emails (V3)]; GetEmails --> ForEach[For each]; subgraph ForeachScope [ ]; ForEach --> HtmlToText[Html to text]; HtmlToText --> AppendToString[Append to string]; end;
```

The flow starts with an 'Initialize variable 2' step, followed by a 'Get messages' step, which then triggers the 'Get emails (V3)' action. This is followed by a 'For each' loop, which contains an 'Html to text' action and an 'Append to string' action.

The emails will be appended to the string "emails" variable along with the URL for each email – this is helpful to verify the original source of the email and to ensure that the AI can reference the email it got the insights on the order delay from.

We will create a foreach loop where we will go through each Outlook message, as the Outlook email is returned in HTML, we will use the action "HTML to text" to parse the HTML and return plain text in response. The input to the "Html to text" action will be the current item in the foreach loop.

After we get the plain text, we will append it to the email's variable along with the URL of the email, follow the format of the URL as bellow or refer to the exported flow on GitHub.

The screenshot shows a Power Automate flow. It starts with a 'Get messages' action, which retrieves messages from an Outlook inbox. This is followed by an 'Append to string variable' action, where the variable name is 'emails'. The 'Value' field of this step contains a dynamic content formula:

```
concat('https://outlook.office.com/mail/inbox/id/', replace(encodeUriComponent(items('For_each')?['conversationId']), '_', '%2B')))
```

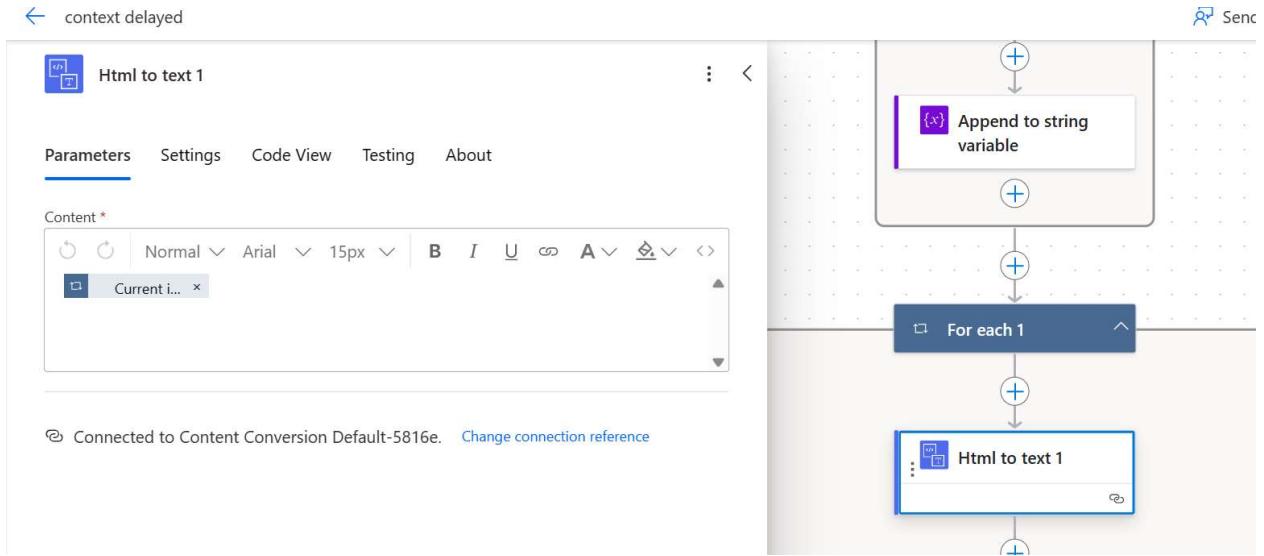
Once we have gotten the emails, we will now work on getting the Teams' messages.

We will create a "For each" action and loop over each message in the for loop, we will check if each message is related to the order by searching for the order number in the Teams message.

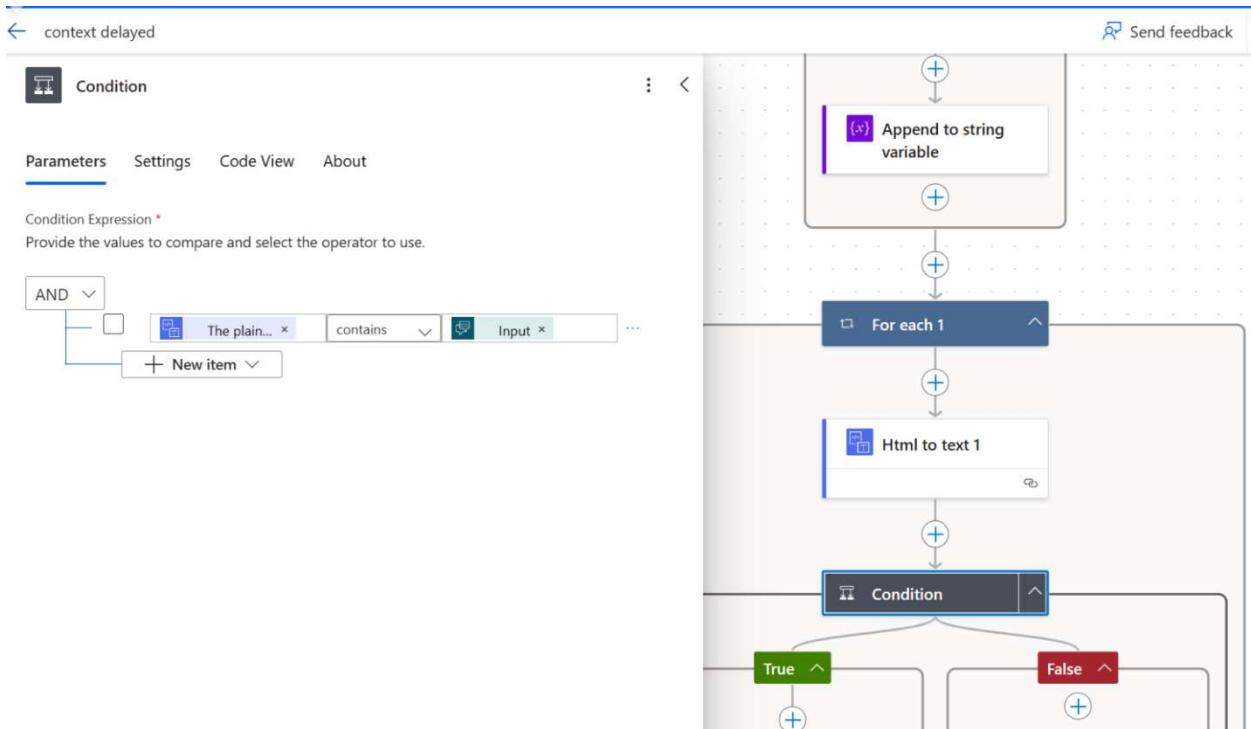
To do this we will loop over all Teams' messages:

The screenshot shows a Power Automate flow. It begins with a 'For each 1' loop. Inside this loop, there is a sequence of steps: an 'Html to text' action, an 'Append to string variable' step (with variable name '{x}'), and another 'For each 1' loop. This inner loop also contains an 'Html to text' action. The 'Select An Output From Previous Steps' dropdown is set to 'Message...', indicating that the output of the first 'For each 1' loop is being used as input for the inner loop.

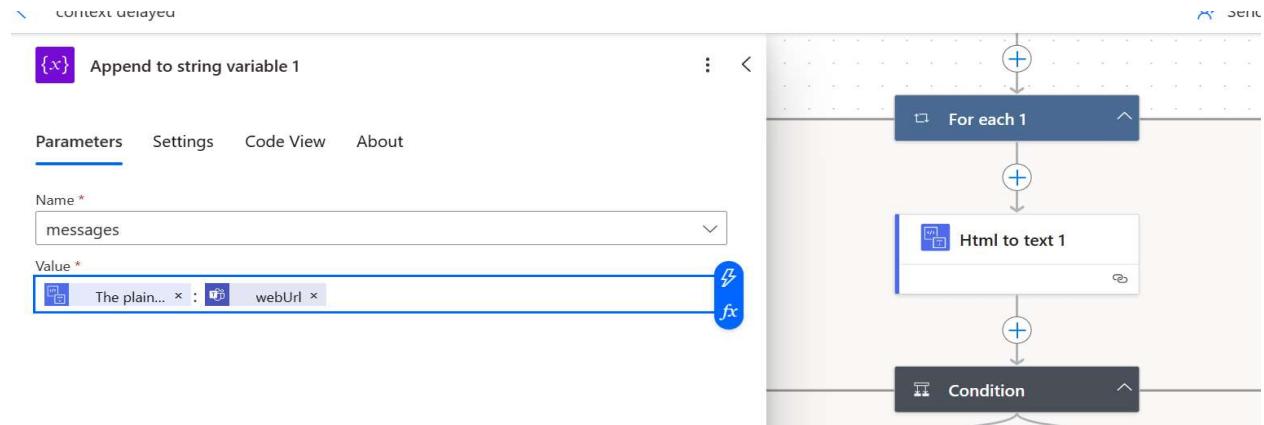
Next we will convert the response from the HTML format to text using the “Html to text” action.



Once we have the Teams message in plain text, we will use the “Condition” action and see if the content contains the order number (gotten from the input variable).



If it does, we will append the message as well as the WebUrl of the Teams message (gotten from the dynamic variables)



After the relevant Teams' messages and the URLs are saved in the variables, we will now send this as an input to Azure OpenAI to summarize and give the salesperson insights on why the order is late. For this we will make a HTTP request to Azure OpenAI as shown below.

The screenshot shows a detailed Power Automate flow. On the left, there is an "HTTP" action with the following configuration:

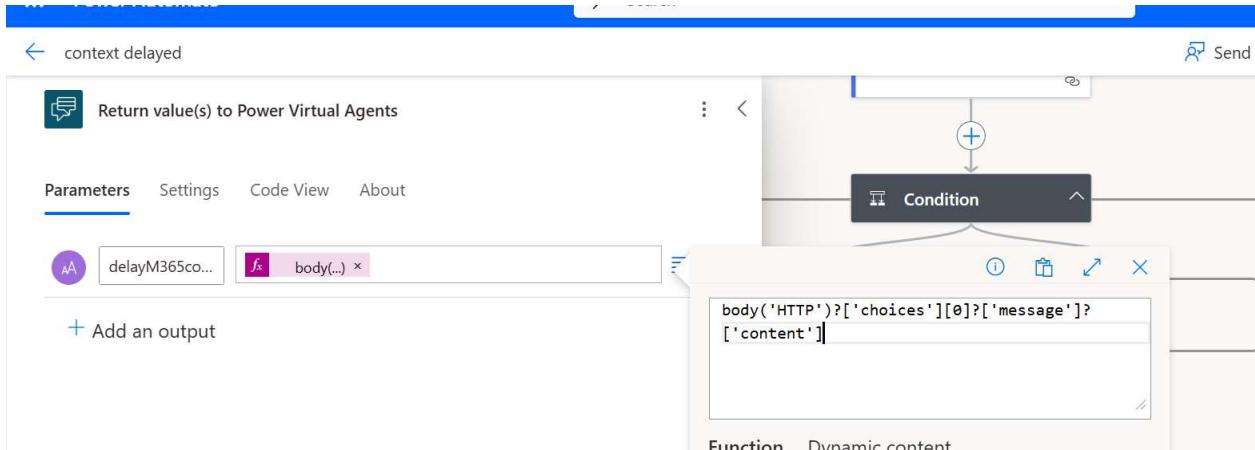
- URI: [REDACTED]
- Method: POST
- Headers:

content-type	application/json
api-key	[REDACTED]
- Queries: Enter key: Enter value
- Body:

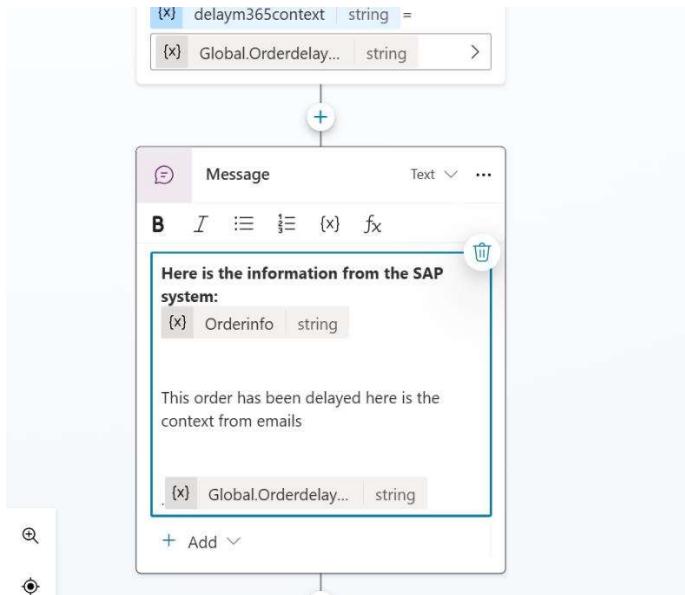
```
{
  "messages": [
    {
      "role": "system",
      "content": "here are some emails and teams messages with information on a order, summarize mentioning which are teams messages and which are emails the reason for the delay: here are the emails: {x} emails {x}, and here are the messages: {x} messages {x}, mention the email as well as the link to the email and if there are no teams messages or no emails mention that Make the URL embedded in a short form text call it link, don't add any of your personal inferences. Also in terms of timestamp, the emails and messages are from new to old, add a blank between each email/team message and seperate them by headings of team message and emails respectively"
    }
  ]
}
```

To the right of the "HTTP" action is a complex logic block. It starts with a "Condition" action branching into "True" and "False" paths. The "True" path contains an "Html to text 1 copy" action, followed by an "Append to string variable 1" action. The "False" path leads to another "Append to string variable 1" action. Both paths converge back to the main flow, which continues with another "HTTP" action and a "Return value(s) to Power Virtual Agents" action.

The output from this will be the parsed response from the HTTP call:



Finally, we will send the response back to Copilot Studio and present this information to the user as shown – the variable “orderinfo” is the output from the “get order status” step.



The next step after giving the order context is to allow the user to ask the bot any question they have about the order on the SAP data from the system.

For this, we will create a Power Automate flow that gets the order status from the SAP system and then feeds it into Azure OpenAI along with the question the user has and Azure OpenAI will interpret the JSON response and answer the question.

The screenshot shows the Microsoft Flow builder interface. At the top, there is a 'Question' step with the title 'Do you want more information on the order?'. Below it is an 'Identify' section with the option 'User's entire response'. Underneath is a 'Save user response as' section with the variable '{x} Global.FullDialog string'. A large blue '+' button is positioned between the two sections. Below this is an 'Action' step titled 'Power Automate inputs (2)'. It contains two inputs: 'FullDialog (String)' and 'docno (String)'. The 'FullDialog' input is set to '{x} Global.FullDialog string'. The 'docno' input is set to '{x} Global.Salesorder... string'. At the bottom of the action step, there is a link 'View flow details'.

Here is the Power Automate flow for this, the name of this from is "Questions on order":

The first step is to add two inputs to this, the document number and the question that the user is asking.

The screenshot shows the Power Automate interface for the "Questions on order" flow. At the top, there's a blue header bar with the title "Power Automate" and a search bar. Below the header, the flow name "Questions on order" is displayed. The main area shows a "When Power Virtual Agents calls a flow" trigger. Under the "Parameters" tab, there are two inputs: "FullDialog" with the value "Please enter your input" and "docno" with the value "Please enter your input". A button "+ Add an input" is visible. To the right, a vertical timeline shows the flow's steps: "Power Virtual Agents calls a flow" followed by "Initialize variable - Chat bot prompt".

The next step is to initialize the prompt variable that we will send to Azure OpenAI.

The screenshot shows the configuration for the "Initialize variable - Chat bot prompt" step in the "Questions on order" flow. The step is identified by the code block placeholder "{x} Initialize variable - Chat bot prompt". Under the "Parameters" tab, the "Name" is set to "System Prompt", the "Type" is "String", and the "Value" is "Enter initial value". The step is positioned in the timeline after the "Power Virtual Agents calls a flow" step and before the "Initialize variable - Chat bot prompt" step.

The next step is to call the SAP function to get order status.

Power Automate

Questions on order

Send feedback

Call SAP function (V2)

Parameters Settings Code View Testing About

SAP System *

API System: [REDACTED]

RFC Name *

BAPI_SALESORDER_GETSTATUS

RfcInputs/SALESDOCUMENT *

docno

Advanced parameters

Showing 2 of 4

Show all Clear all

RFC Group Filter

*

Auto Commit

No

Power Virtual
is calls a flow

Create variable -
bot prompt

AP function

Create variable -
Add data to chatbot

```
graph TD; A[Power Virtual is calls a flow] --> B[Create variable - bot prompt]; B --> C((AP function)); C --> D[Create variable - Add data to chatbot]
```

Power Automate

Questions on order

{x} Set variable - Add SAP data to chatbot prompt

Parameters Settings Code View About

Name * System Prompt

Value *

here is the data on the sales document answer any questions on the status information by understanding the JSON given here and any follow up questions the user has as well.
Here is the information.

fx string(...)
Dont mention it is from JSON data but mention the data is from the SAP System.

Power Virtual
This calls a flow

```
string(body('Call_SAP_function_(V2)')?['STATUSINFO'])
```

Function Dynamic content

Search

fx String functions See More

concat(text 1 text 2)

Next, we will add the user's question and the SAP JSON response to the HTTP Request to Azure OpenAI.

Power Automate

Questions on order

{x} Initialize variable for request to bot

Parameters Settings Code View About

Name * messages

Type * Array

Value

```
[{"role": "system", "content": "System P..."}, {"role": "user", "content": "FullDialog"}]
```

MSFT (default)

Send feedback Copilot Save Flow checker Test New de

```

graph TD
    A[Set variable - Add SAP data to chatbot prompt] --> B[Initialize variable for request to bot]
    B --> C[HTTP]
    C --> D[Initialize variable - reply content]
  
```

Questions on order

Send feedback Copilot Save Flow checker

HTTP

Parameters Settings Code View About

URI: [REDACTED]
Method: POST
Headers:
content-type: application/json
api-key: [REDACTED]
Queries:
Enter key Enter value
Body:
{
 "messages": [{ }]
}
Cookie:
Enter HTTP cookie

```

graph TD
    A[Set variable - Add SAP data to chatbot prompt] --> B[Initialize variable for request to bot]
    B --> C[HTTP]
    C --> D[Initialize variable - reply content]
    D --> E[Return value(s) to]
  
```

Lastly, we will process the response, store it in a variable, and send it back to the Copilot Studio – this variable will be called "Generatedanswer". (Note: the value for the response is the same formula as the previous Power Automate flow: body('HTTP')['choices'][0]?['message']['content'])

Questions on order

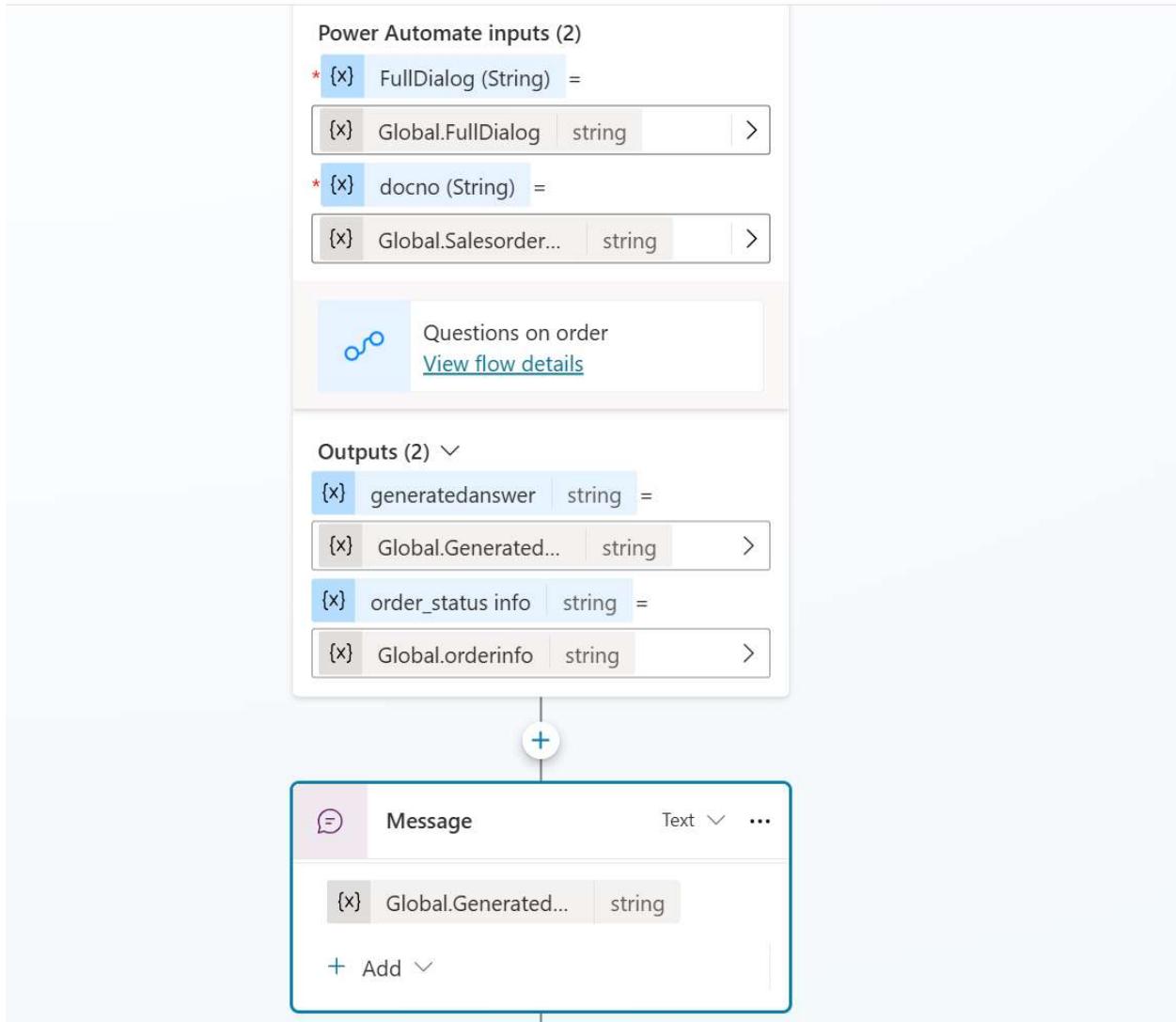
Return value(s) to Power Virtual Agents copy

Parameters Settings Code View About

Generatedan... fx body(...)
order status i... fx string(...)

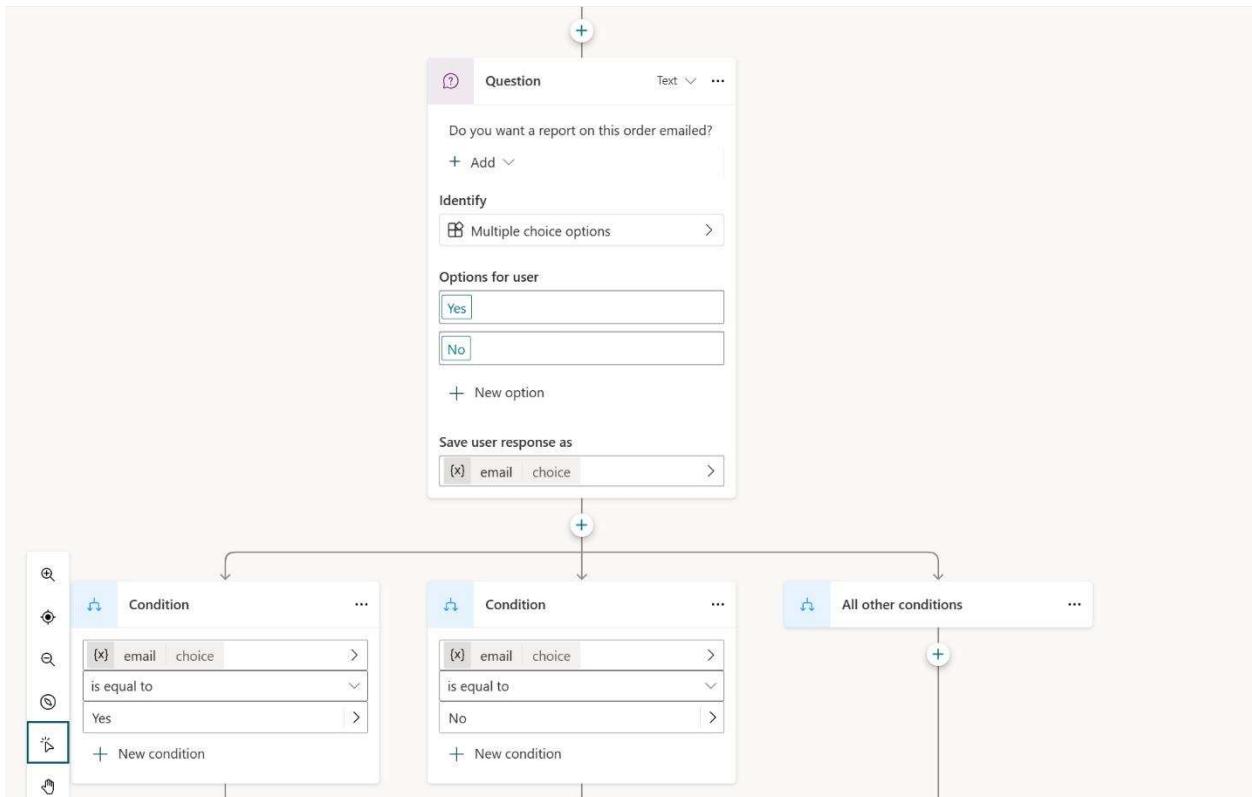
+ Add an output

The other variable we will send back is the order status information we got from the SAP System. This variable could be used by other potential topics, and we save it as a global variable in the Copilot Studio so that it can be used across topics.

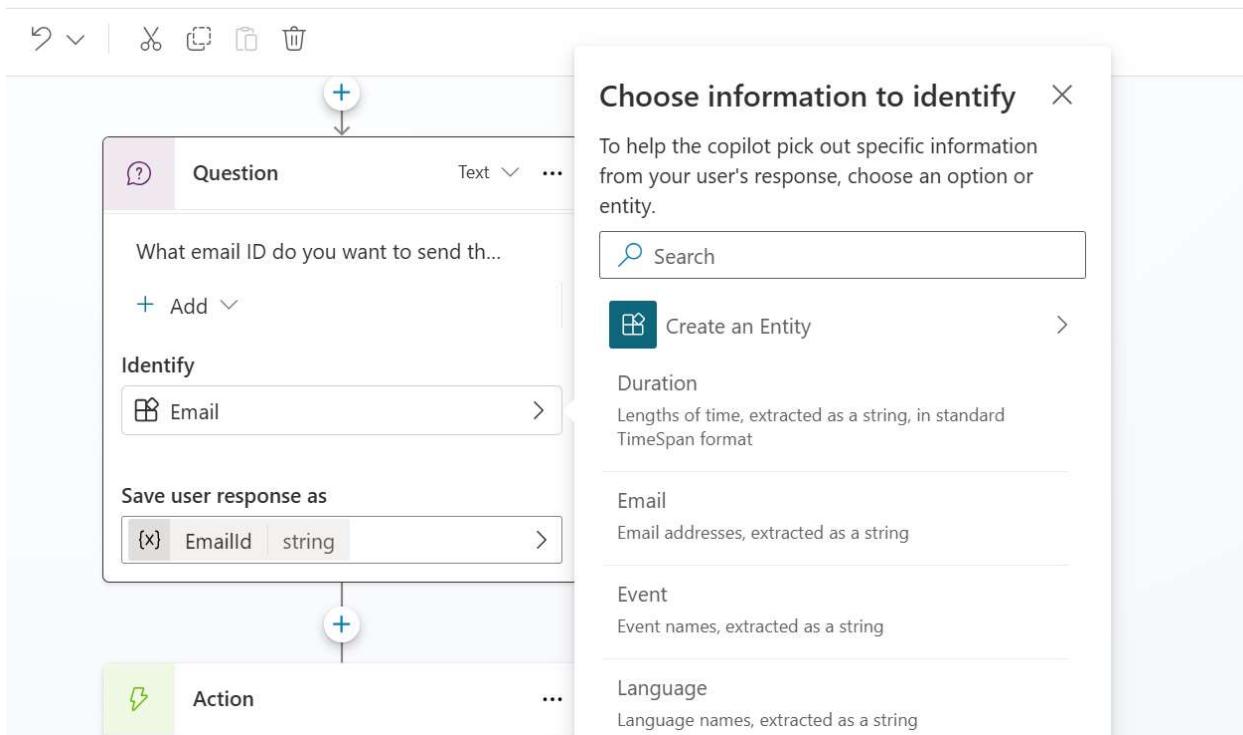


Lastly, if the salesperson wants to get back to the customer with the analysis, we will ask the salesperson if they want the entire conversation/ questions they asked about the order to be summarized by AI and put into a report to be sent to the customer (or anyone via Outlook).

For this back in the Copilot Studio, we will ask the user a question as shown here:



If the user responds that they want to send an email, we will ask them for the email ID they want to send this to, for this, we need to ask another question and select the readymade email entity as the response (as we did for customer number and sales order number earlier).



Finally, after that, we will call the final Power Automate flow with the variables shown below. This Power Automate flow is called "report email"

This flow will combine the responses from the "context delayed" flow (the summary of the Teams' chats and Outlook emails) as well as the recent response from Azure OpenAI and send a request to Azure OpenAI to respond in an email format

The screenshot shows the 'Power Automate inputs (4)' section and the 'Outputs (1)' section of a flow configuration.

Power Automate inputs (4)

- * {x} Delay context (String) = {x} Global.Orderdelay... string >
- * {x} order info SAP (String) = {x} Global.Generated... string >
- * {x} order no (String) = {x} Global.Salesorder... string >
- * {x} email id (String) = {x} Emailld string >

Outputs (1) ▼

- {x} email_has been se... string = {x} Emailhasbeensent string >

A callout box highlights the 'report email' step, which includes a link to 'View flow details'.

The steps for this flow are:

The first step is to define the 4 variables we need. The summary of Outlook + Teams messages, the response to the previous question asked from Azure OpenAI, the order number and finally the email ID we want to send the email to – this would be the customers email ID.

Power Automate

report email

When Power Virtual Agents calls a flow

Parameters Settings Code View About

Delay context Report order no 1 email id

Add an input

```

graph TD
    A[When Power Virtual Agents calls a flow] --> B[HTTP]
    
```

Next we combine all these inputs and send a request to Azure OpenAI to generate an email out of it with formatting we want as shown here:

HTTP

Parameters Settings Code View About

URI *

Method * POST

Headers

content-type	application/json
api-key	<input type="text" value="REDACTED"/>

Queries

Enter key	Enter value
-----------	-------------

Body

```
{
  "messages": [
    {
      "role": "system",
      "content": "Take this information from there inputs and create formal business letter formatted professionally in HTML formatting on why the order was delayed .Also format the letter in a professional manner and return the response in bullet format and add an empty line between each bullet point in HTML formatting. The heading should be in bold in html formatting. Start the email off with dear sir/Madam and no text/description above it. and then start the rest of the email off with heading in bold with HTML formating. Here is the input (do not include links)  +  . For the signing off , just end with Thank you, Noopur "
    }
  ]
}
```

Cookie

Enter HTTP cookie

```

graph TD
    A[When Copilot Studio calls a flow] --> B[HTTP]
    B --> C[Send an email V2]
    C --> D[Return value(s) to Power Virtual Agents]
    
```

Next we use the outlook send email function to send the email to the customer and populate the fields as shown below.

The screenshot shows the 'Send an email (V2)' configuration screen on the left and the flow editor on the right.

Send an email (V2) Configuration:

- Parameters:** To *: email id, Subject *: Report on order [order no] delay, Body *: body(...)
- Advanced parameters:** Showing 1 of 7
- Importance:** Normal

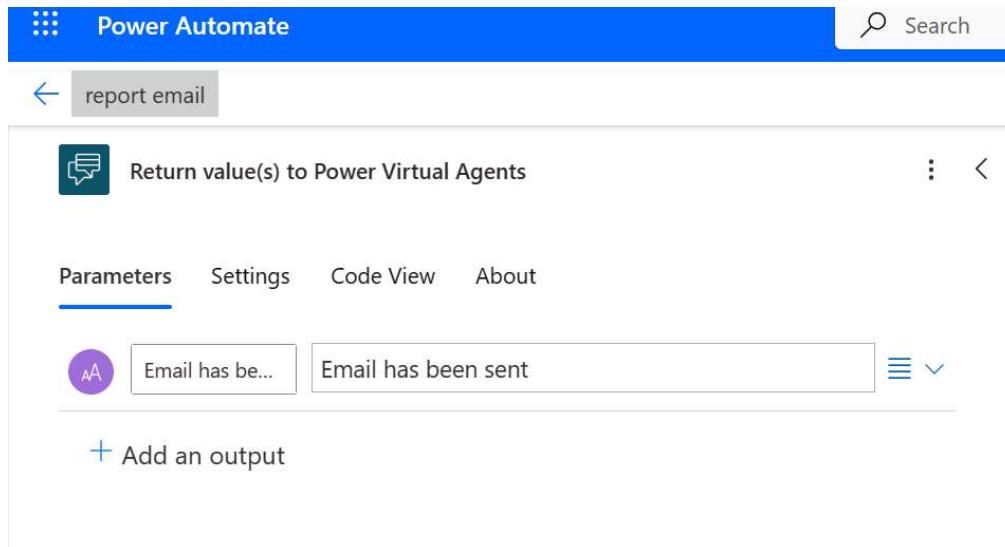
Flow Editor (Power Automate):

```
graph TD; A[When Power Virtual Agents calls a flow] --> B[HTTP]; B --> C[Send an email (V2)]; C --> D[Return value(s) to Power Virtual Agents]
```

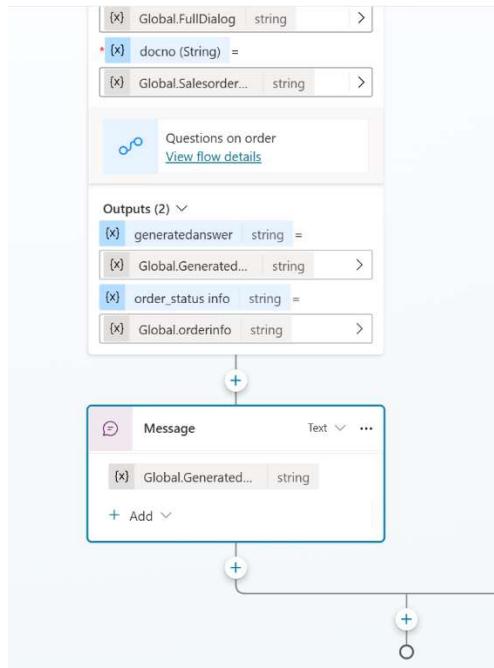
The flow starts with 'When Power Virtual Agents calls a flow', followed by an 'HTTP' step, then the 'Send an email (V2)' step, and finally 'Return value(s) to Power Virtual Agents'.

The body of the email here is the response from Azure OpenAI (which is the generated email in this case) - this is the expression for it as used before body
“('HTTP')?['choices'][0]?['message']?['content']”.

Finally we send a response back to the Copilot Studio stating that the email has been sent.



The chat bot sends a final response to the salesperson informing that the email has been sent.



This concludes the flow.

You can now save and Publish the Copilot by clicking it here:

A screenshot of the SAP Order Status Copilot Overview page. At the top, there is a navigation bar with tabs: Overview, Knowledge, Topics, Actions, Analytics, and Channels. To the right of the tabs, it says "Published 5/30/2024". Below the navigation bar, there is a large central area with some placeholder text and images. At the bottom right of this area, there is a prominent blue "Publish" button.

You can now finally launch it on teams by navigating to the Channels tab and clicking on Teams.

A screenshot of the SAP Order Status Copilot Channels page. At the top, there is a navigation bar with tabs: Overview, Knowledge, Topics, Actions, Analytics, and Channels. The "Channels" tab is currently selected and highlighted in blue. Below the navigation bar, there is a section titled "Channels" with the sub-instruction "Configure your copilot channels to meet your customers where they are." There are six cards arranged in two rows of three, each representing a different channel:

- Microsoft Teams**: Represented by a blue icon with a white "T".
- Demo website**: Represented by a blue icon with a white magnifying glass.
- Custom website**: Represented by a grey icon with a white globe.

- Mobile app**: Represented by a purple icon with a white smartphone.
- Facebook**: Represented by a blue icon with a white "f".
- Skype**: Represented by a light blue icon with a white "S".

As mentioned, this scenario is just a starting point will give you all the raw materials and components to build more scenarios in different domains.