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.

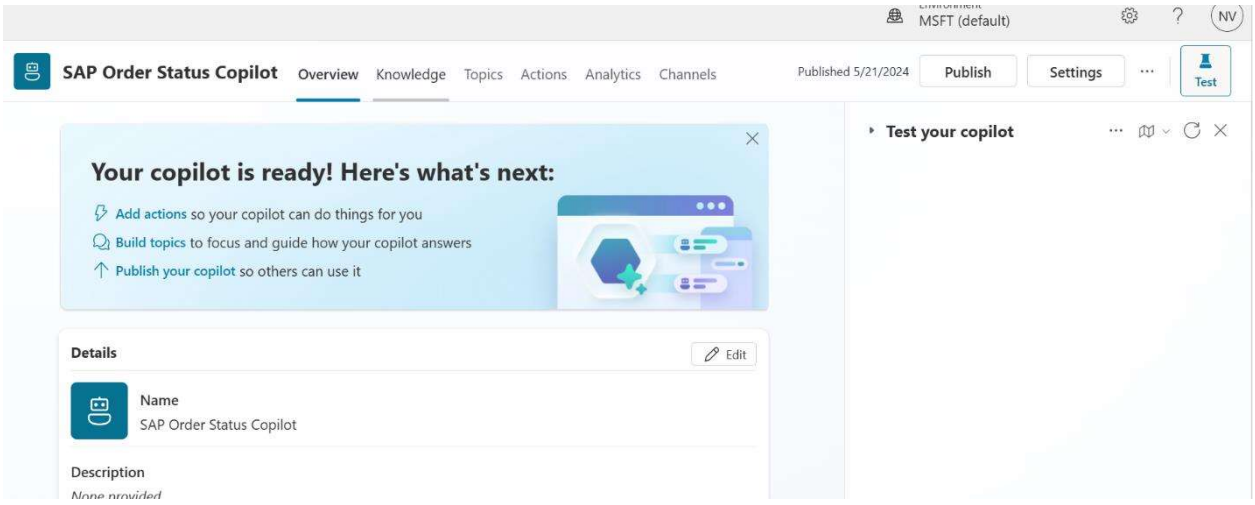
    - Power Platform + SAP - Installing the On-premises Data Gateway
    - Power Platform + SAP - Calling "Remote Function Call" (RFC) in SAP (youtube.com)

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.

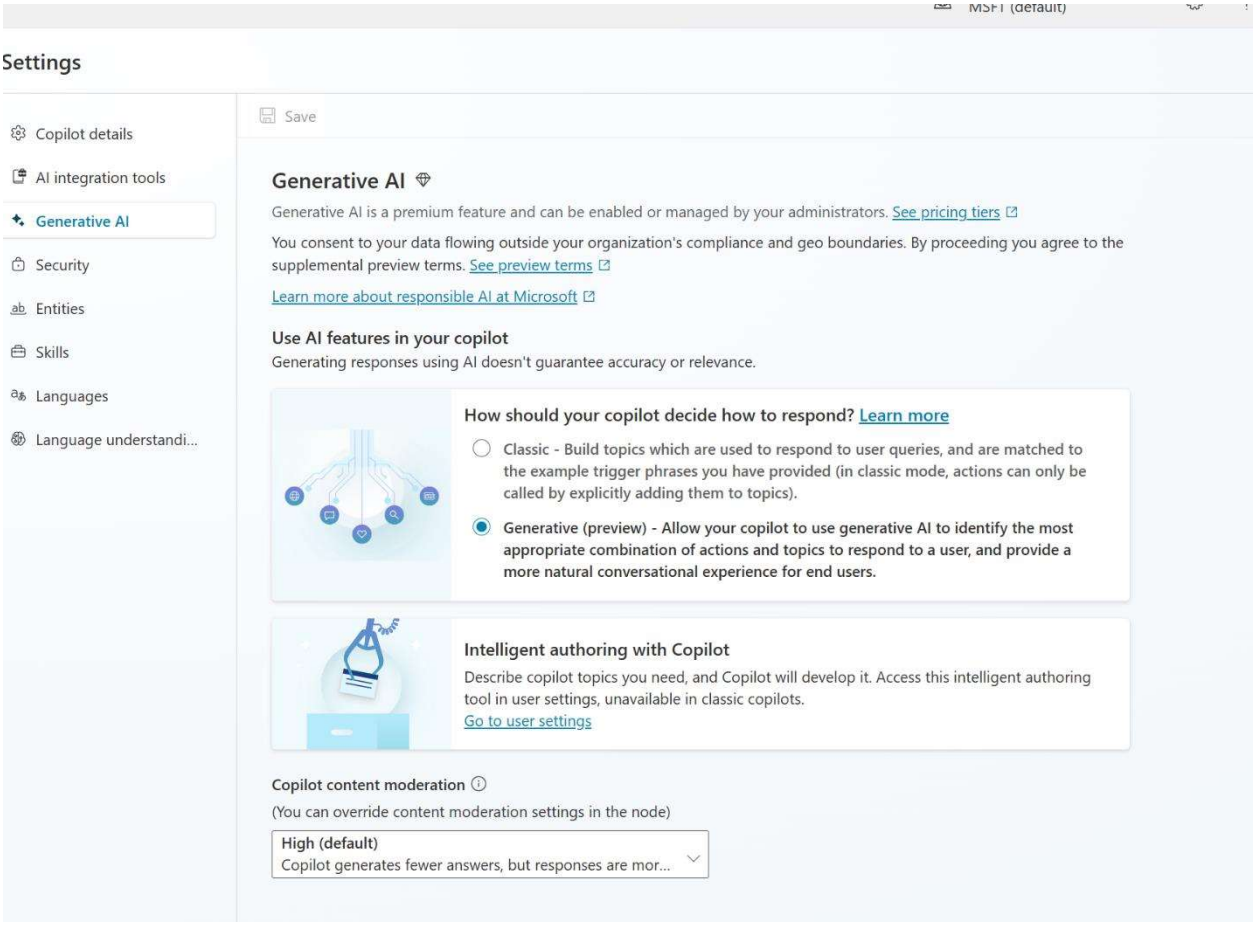
Let us begin by getting started with the Copilot Studio:

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



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



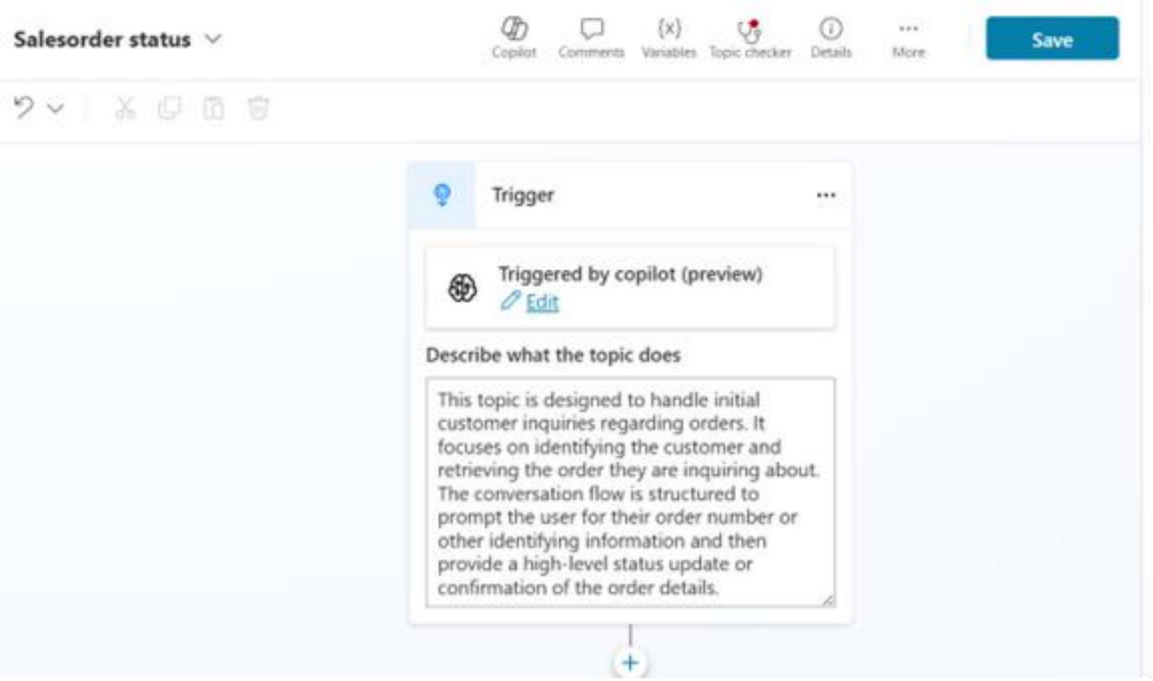
The next step is to create a new topic and the name is – “Salesorder status” – this is the topic that the salesperson will use to get the order status and information on the order.

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!

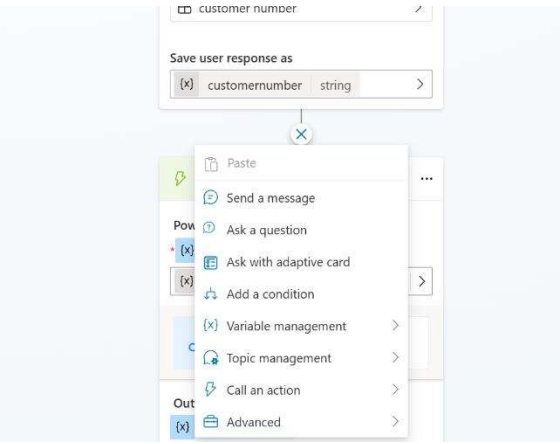
Therefore, the first step after creating a new topic is to write a description of the topic. This is what will help the AI identify what this flow is for.

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]

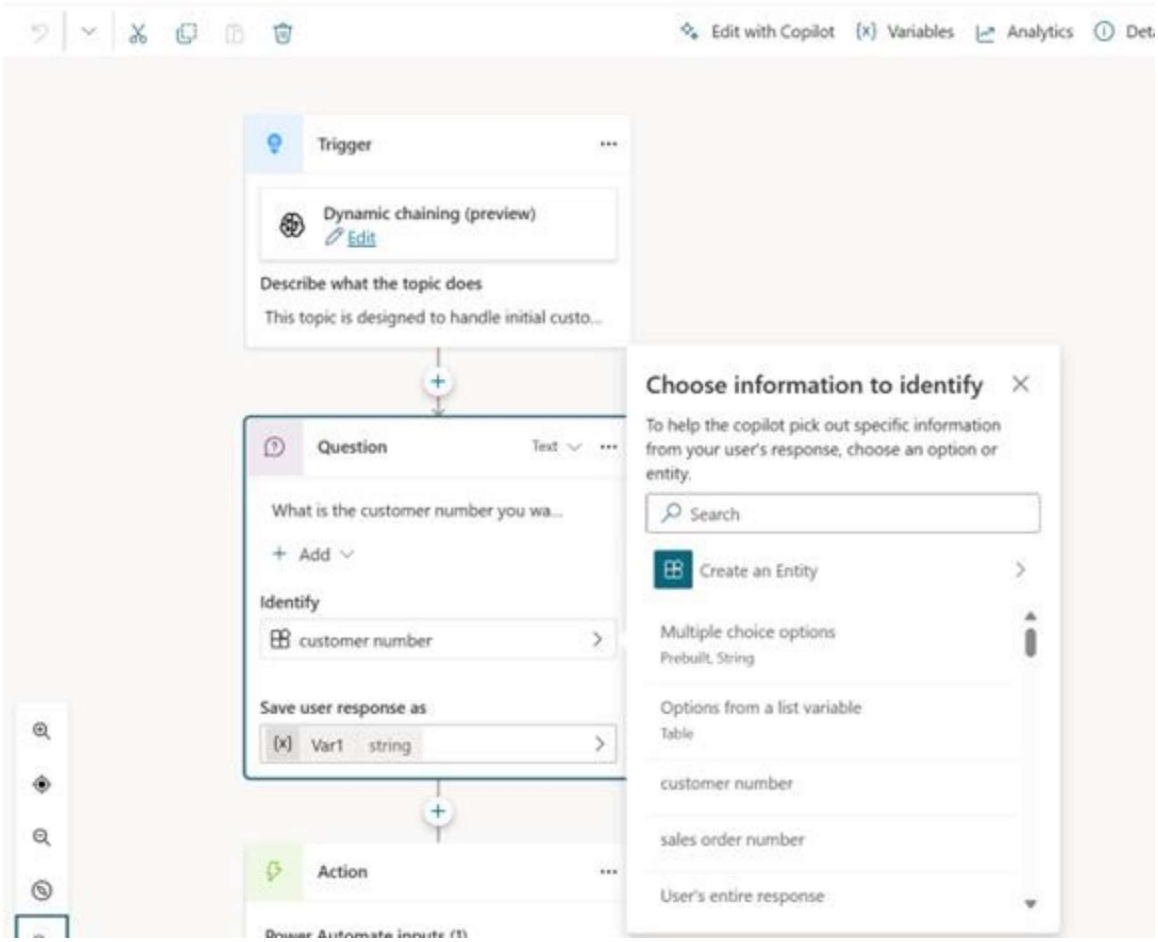
Here is the image of the topic description used for this. (Fun fact- Copilot was used to write the description)



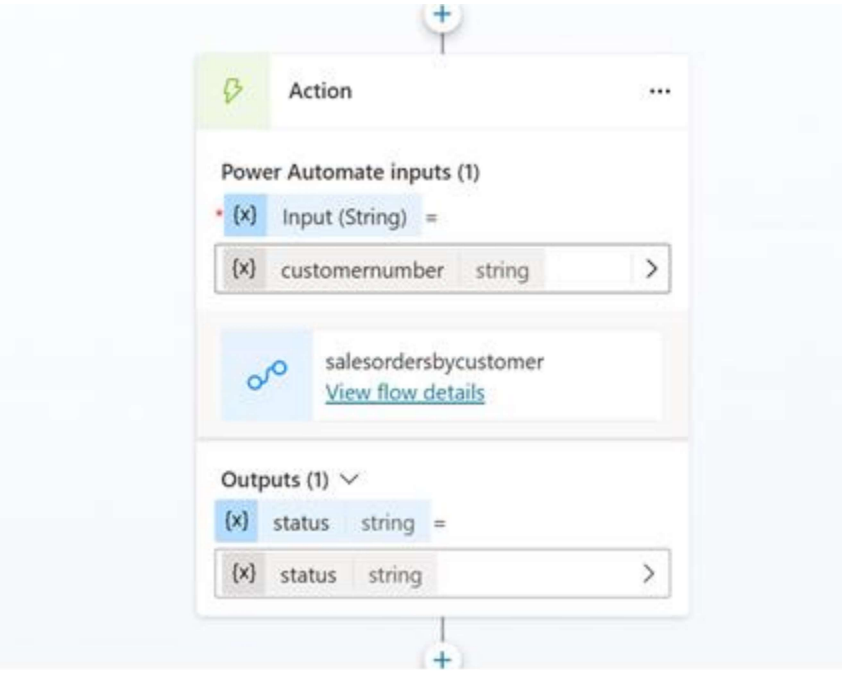
The next step is to add a "question node" to ask the user what customer number they want this information for.



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.

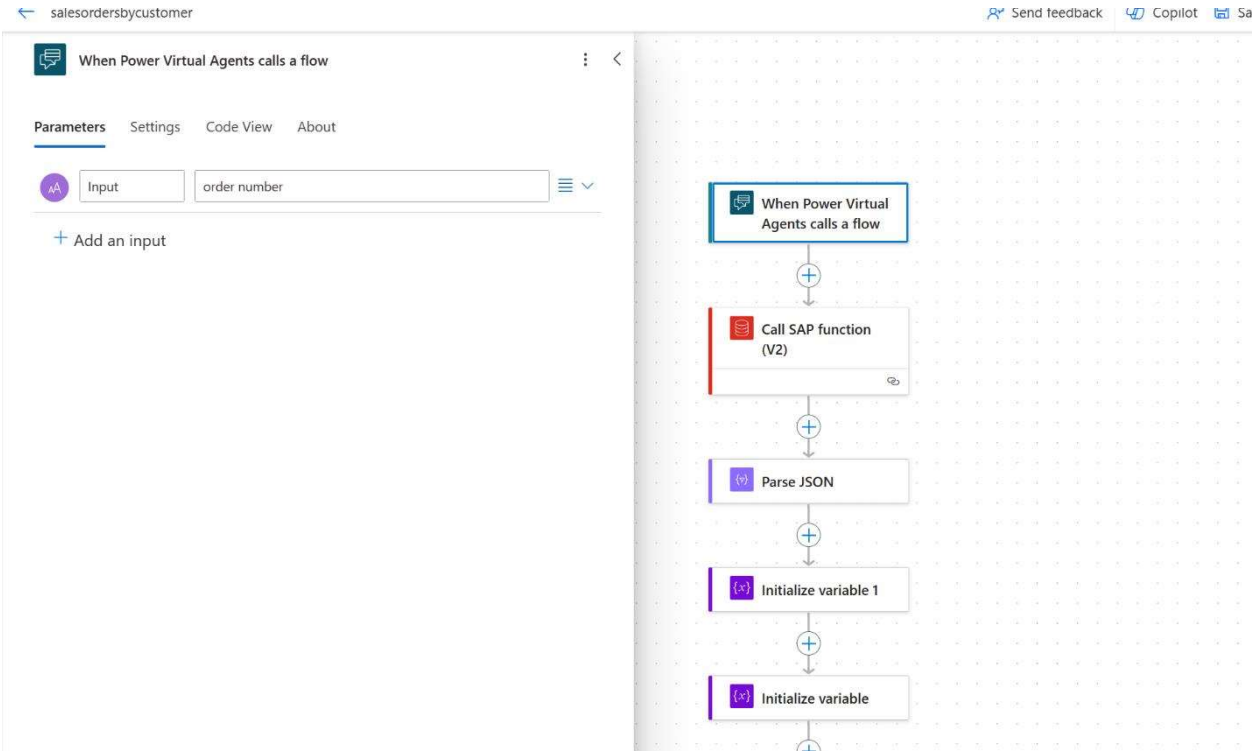


The next step is to now call an action and create a Power Automate flow that gets and parses this information from the SAP system.

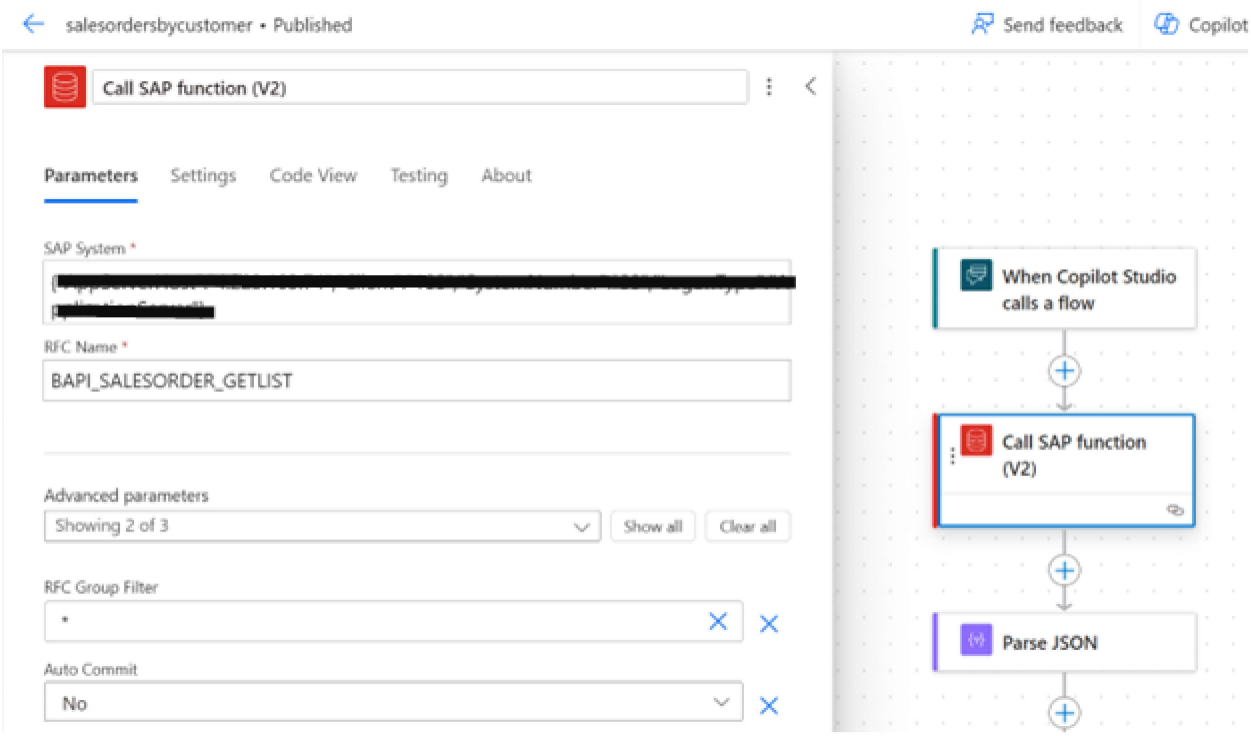


Below is the description of the first Power Automate flow to get all the orders for each customer with each step explained. This flow is called “salesorderbycustomer”.

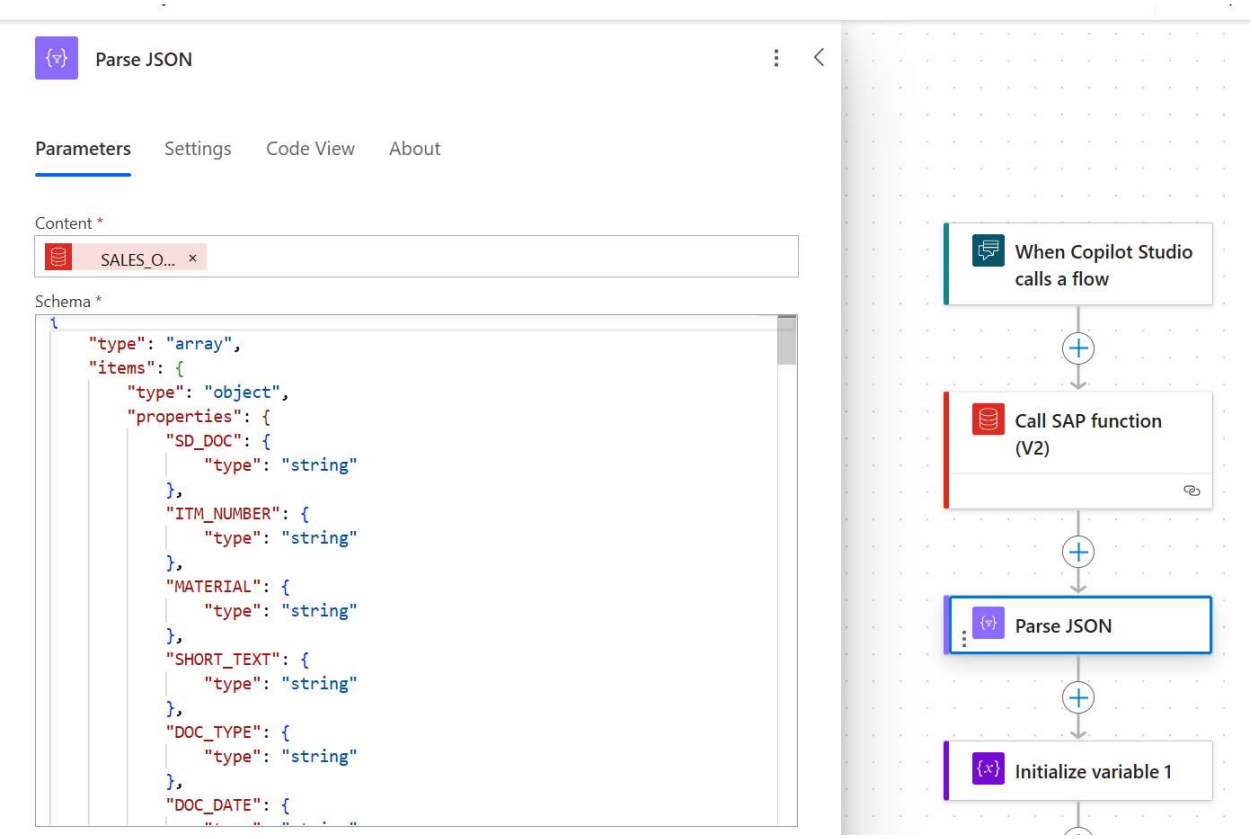
The input to the Power Automate flow will be the customer number extracted from the input from the user.



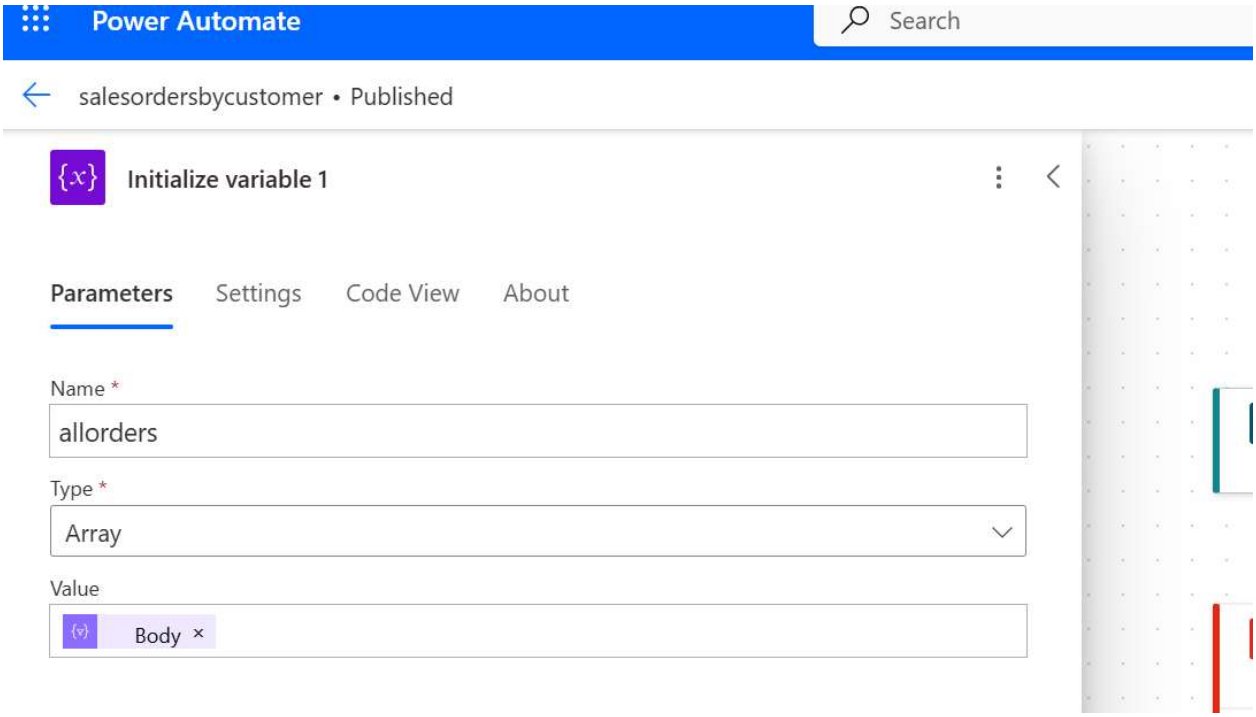
The next step will be to call the SAP BAPI (BAPI_ SALESORDER_GETSTATUS). For this, you must provide the customer number as well as the sales organization number. You would also have to enter your SAP server credentials (as shown in the SAP set up videos linked above in the prerequisites.).



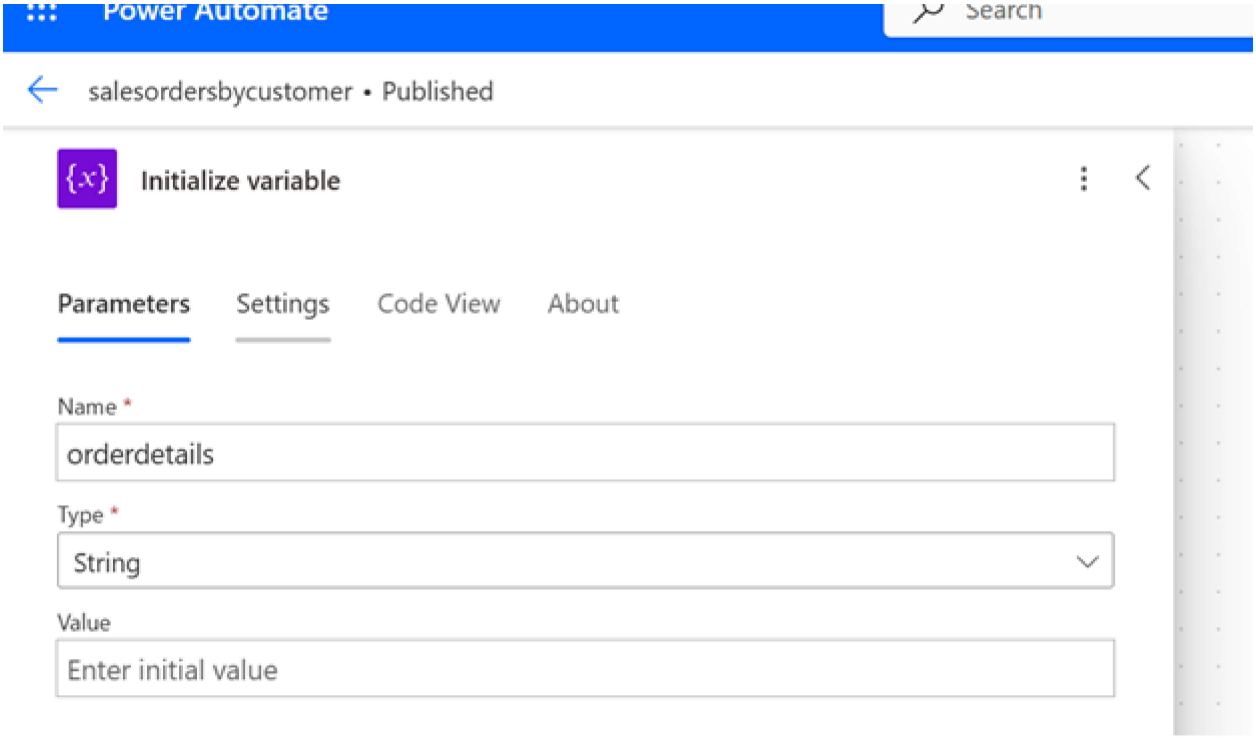
Once you call the SAP BAPI, you will get a JSON response. This response needs to be parsed and stored so that we can use the information to perform further tasks. To do this, you must provide a sample JSON. The JSON is available on GitHub with the exported Power Automate flow.



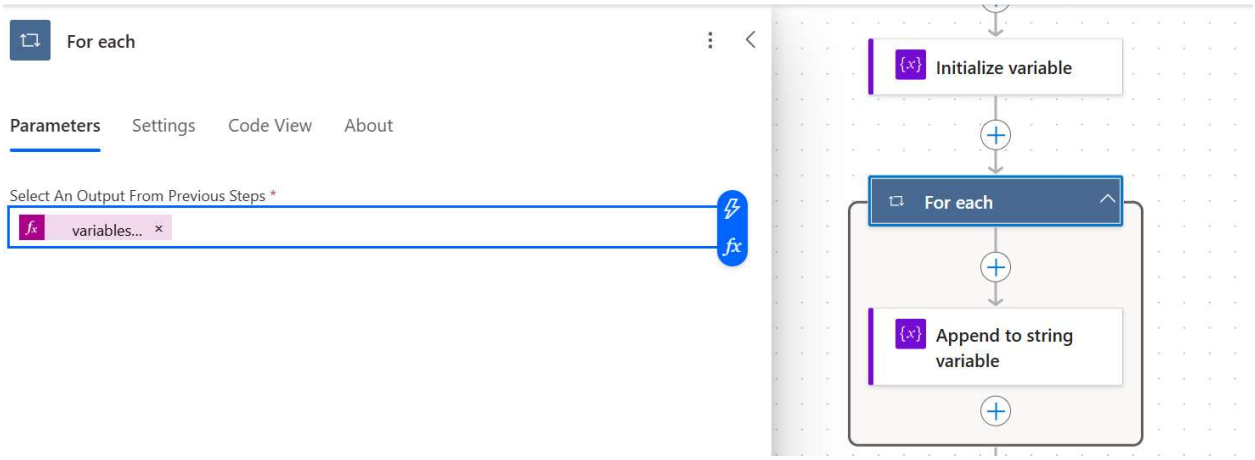
The next step is to initialize two variables – one called "allorders" which has the type "array", and its values will be the body of the JSON we retrieved previously.



The 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.”

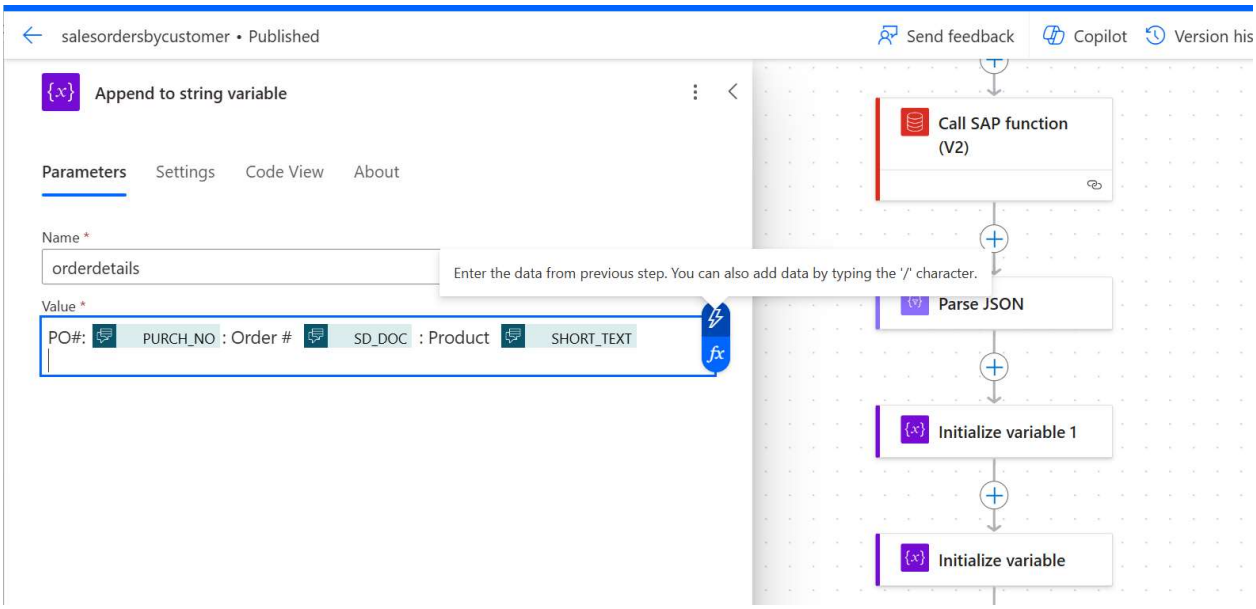


Next, we need to loop over all the sales documents of the customer, go over each and then take all the fields we need from it, and finally paste it in the response. For this, we will use the for each loop and go over the variable “allorders”.

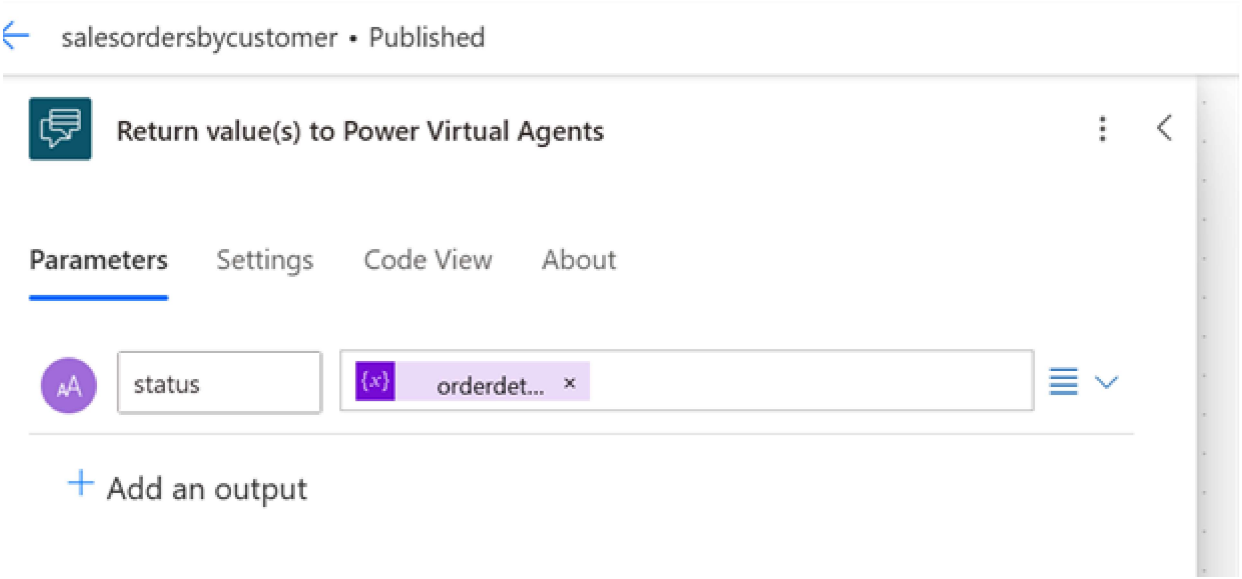


Inside the foreach loop, we will append the fields as shown below to the variable "orderdetails".

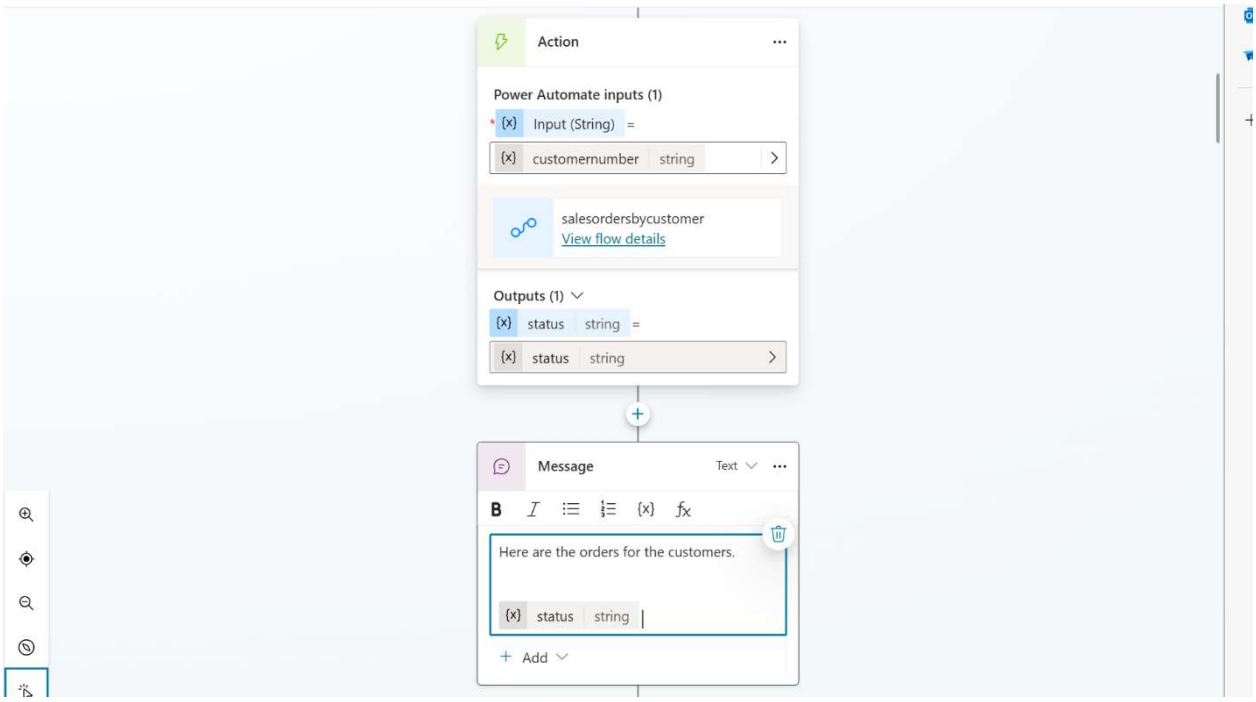
You can add these dynamic fields from the dynamic variables shown below.



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.

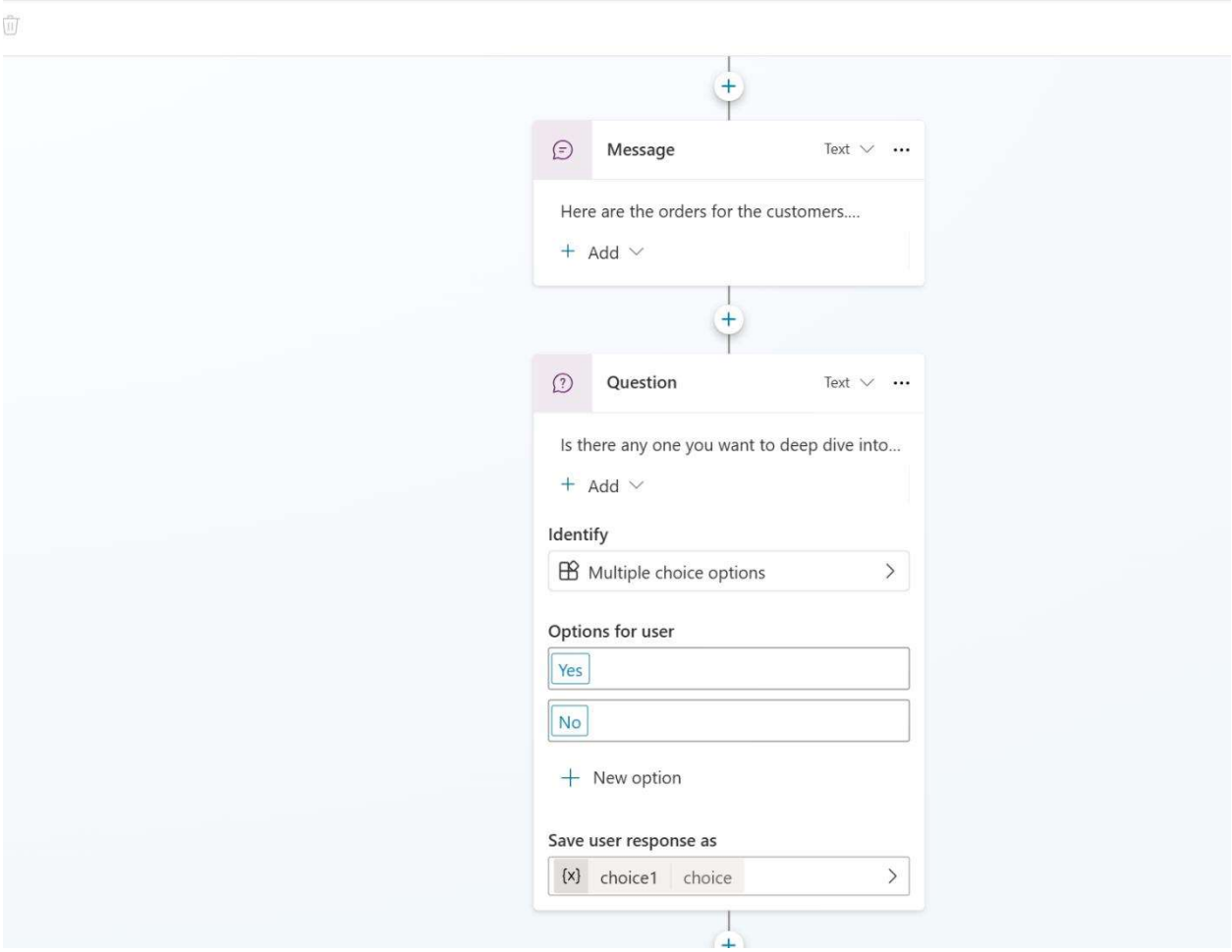


Now back in the Copilot Studio we will append the response returned by the Power Automate flow as shown like this.

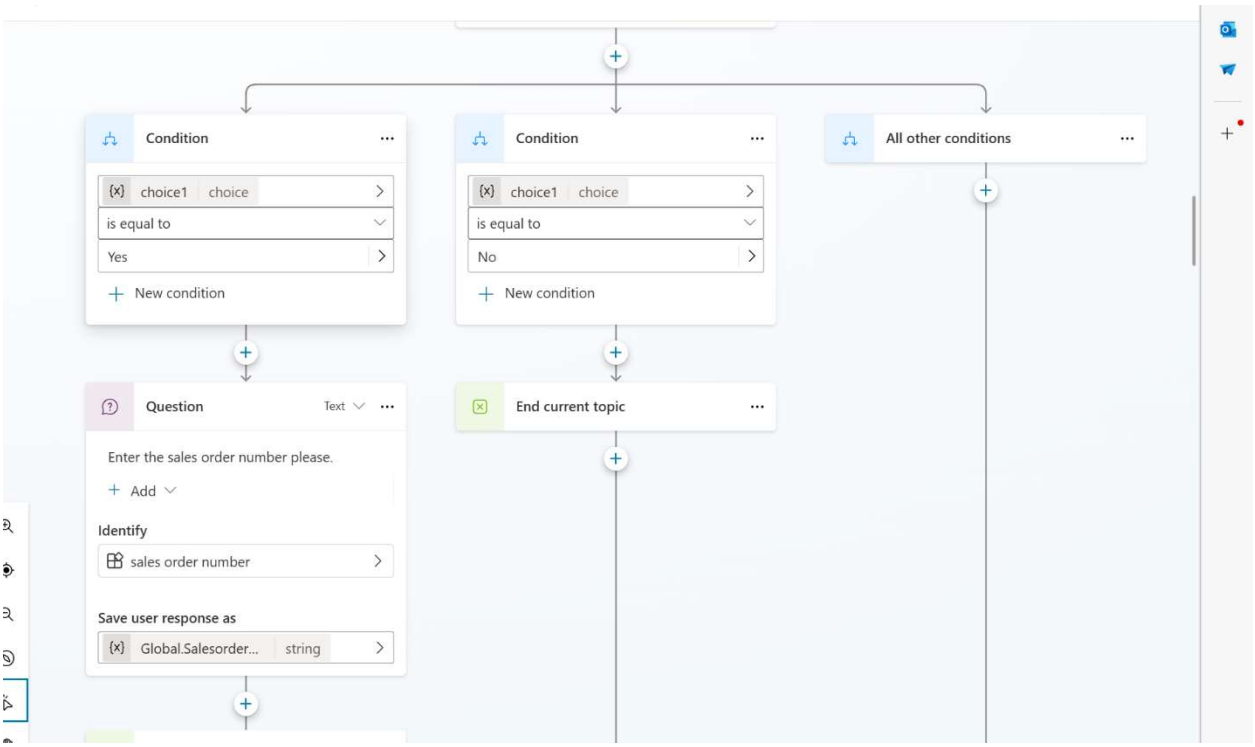


Once we have shown the salesperson all the customers' sales orders, they can now act on it and deep dive into the sales order they want to look into.

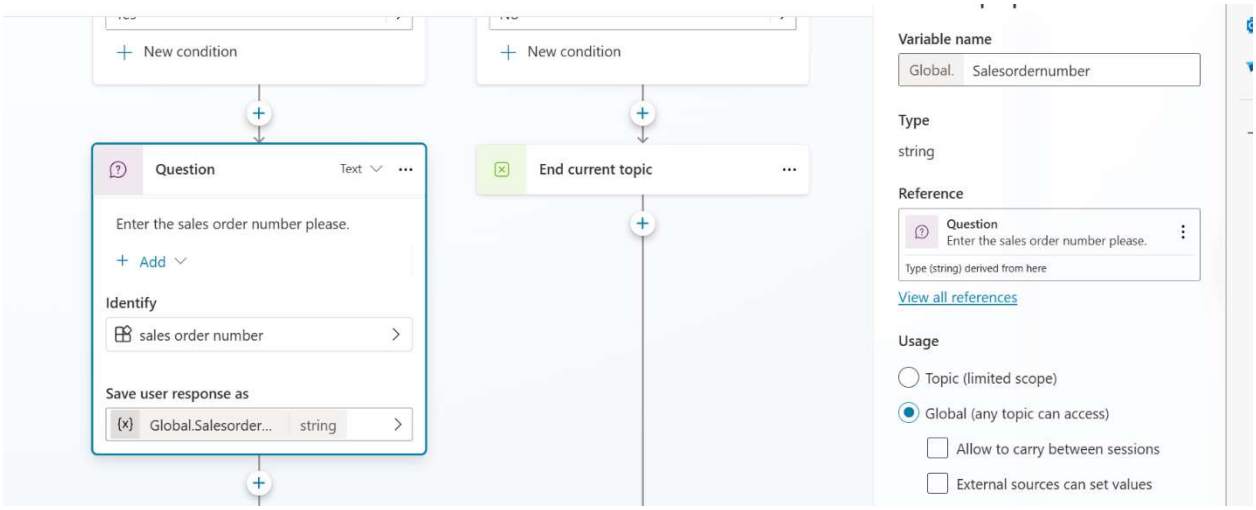
Note: In this demonstration one such action that the salesperson could take has been shown. The action we have selected for this demo is asking the salesperson if there is any particular problematic order they want to dive into and confirm the status of. However, you could create any such flow based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



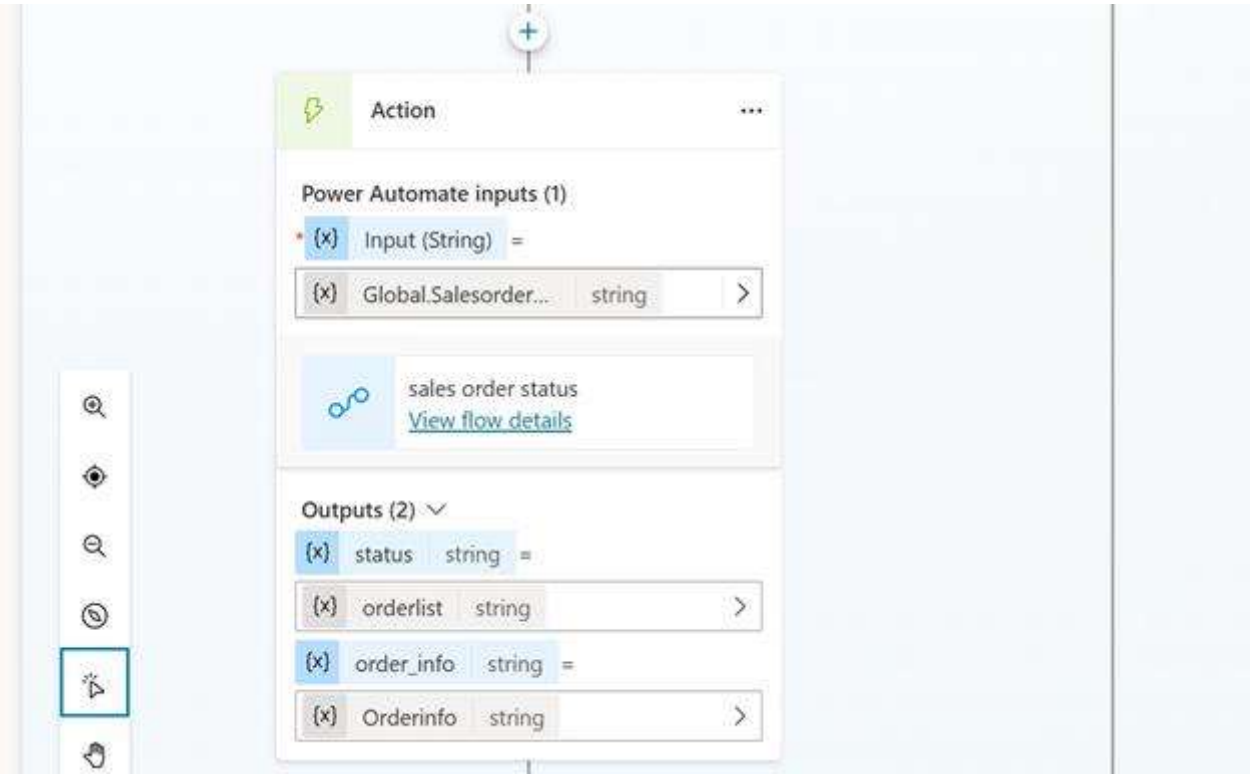
We will create a question node and ask the salesperson to enter the order number in the Copilot studio. We will once again create a new entity with the regex in the format of the SAP sales order number (the same way we did it for the customer number above).



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.



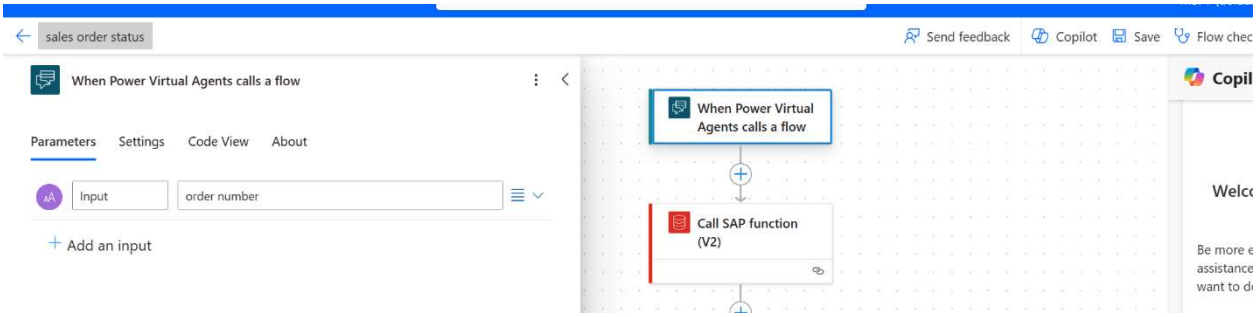
Once we have the sales order number that the customer wants to use, we will now dive deep into this sales order and find out about its status from the SAP system and find out whether it is delayed or not.

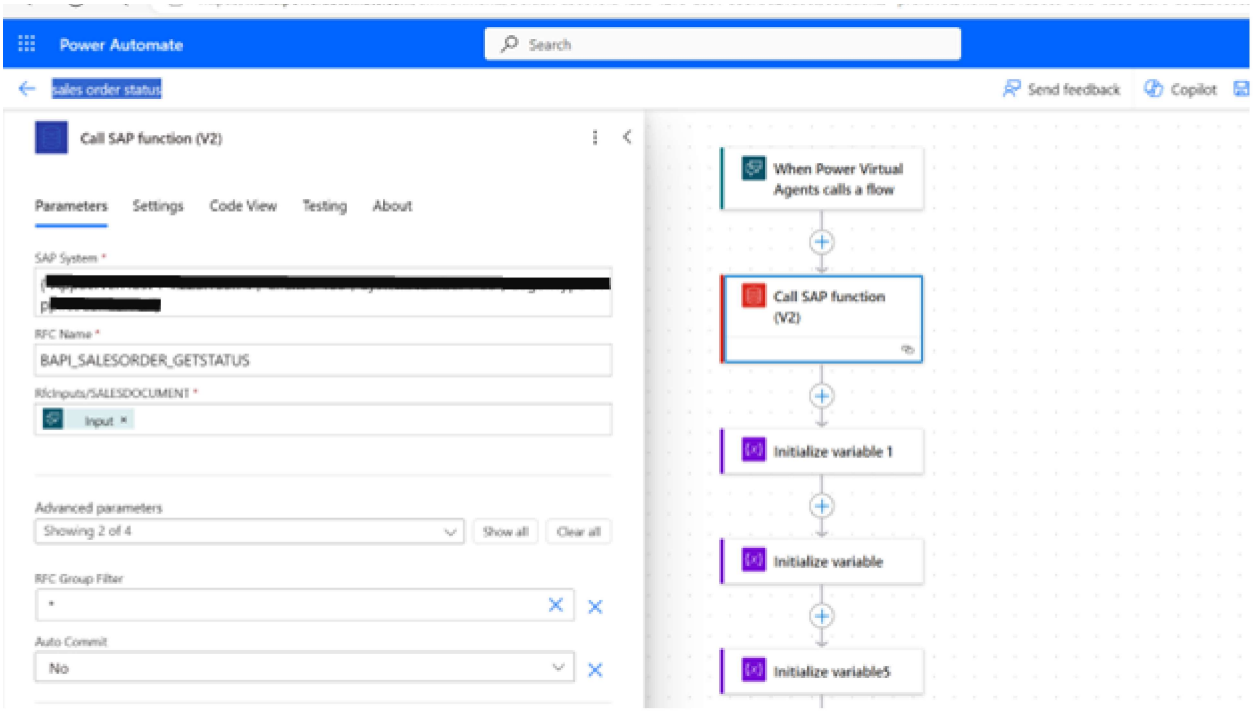


Therefore, the next Power Automate flow we need to call is one that gets the order status for the order the customer mentioned and returns a response on whether the order is delayed or on time. The input to the flow will be the sales order number.

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

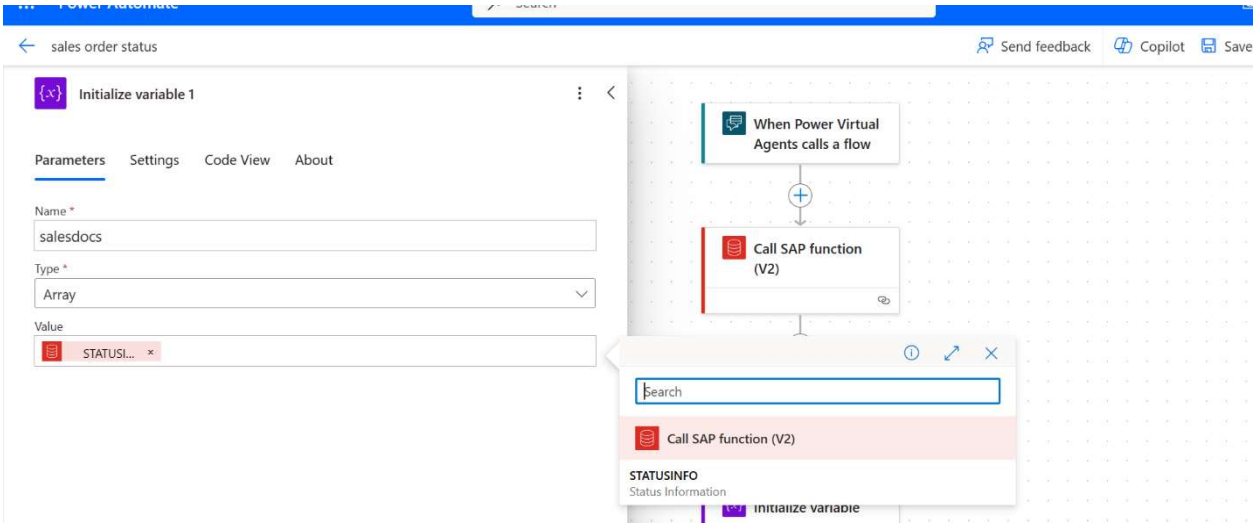
We will first take the order number as an input to the Power Automate flow, next, we call the BAPI_SALESORDER_GETSTATUS – this is to get the order status information for the sales order.

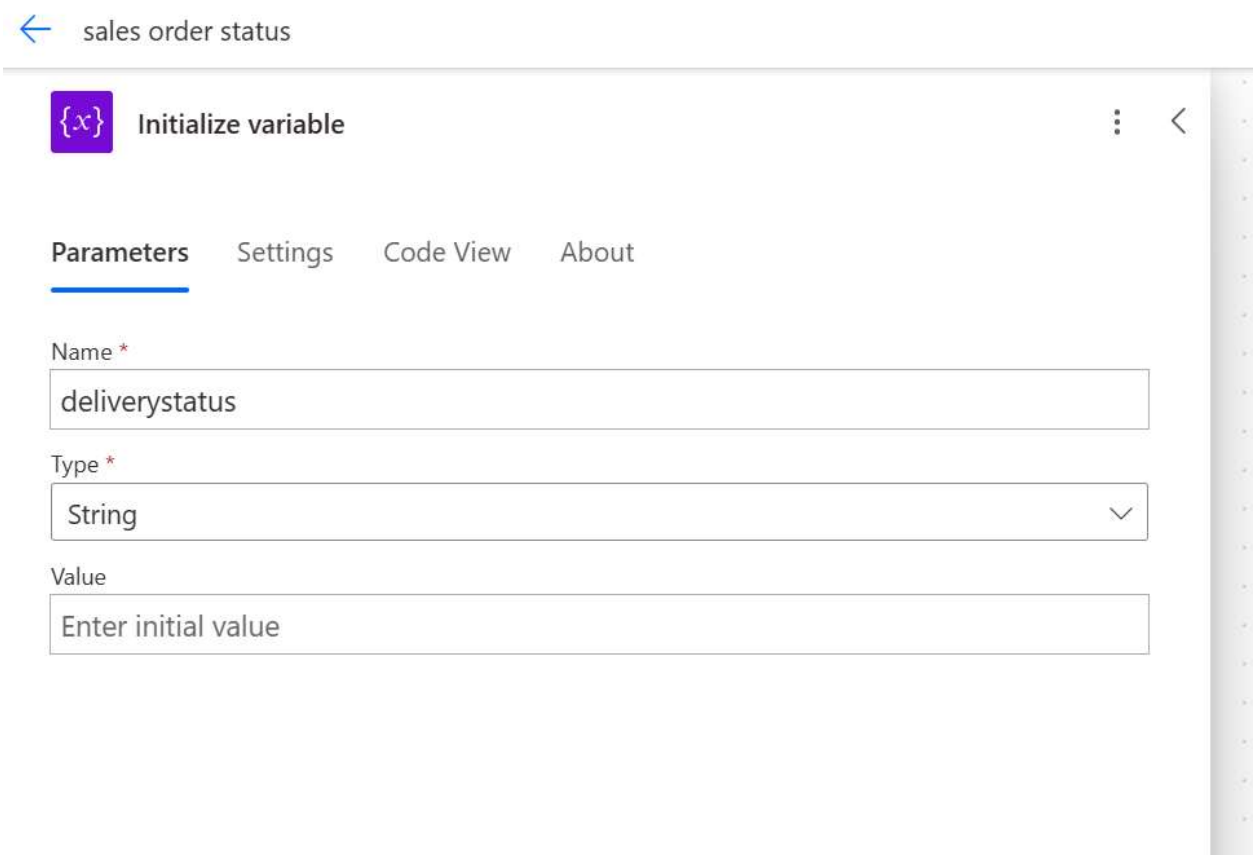




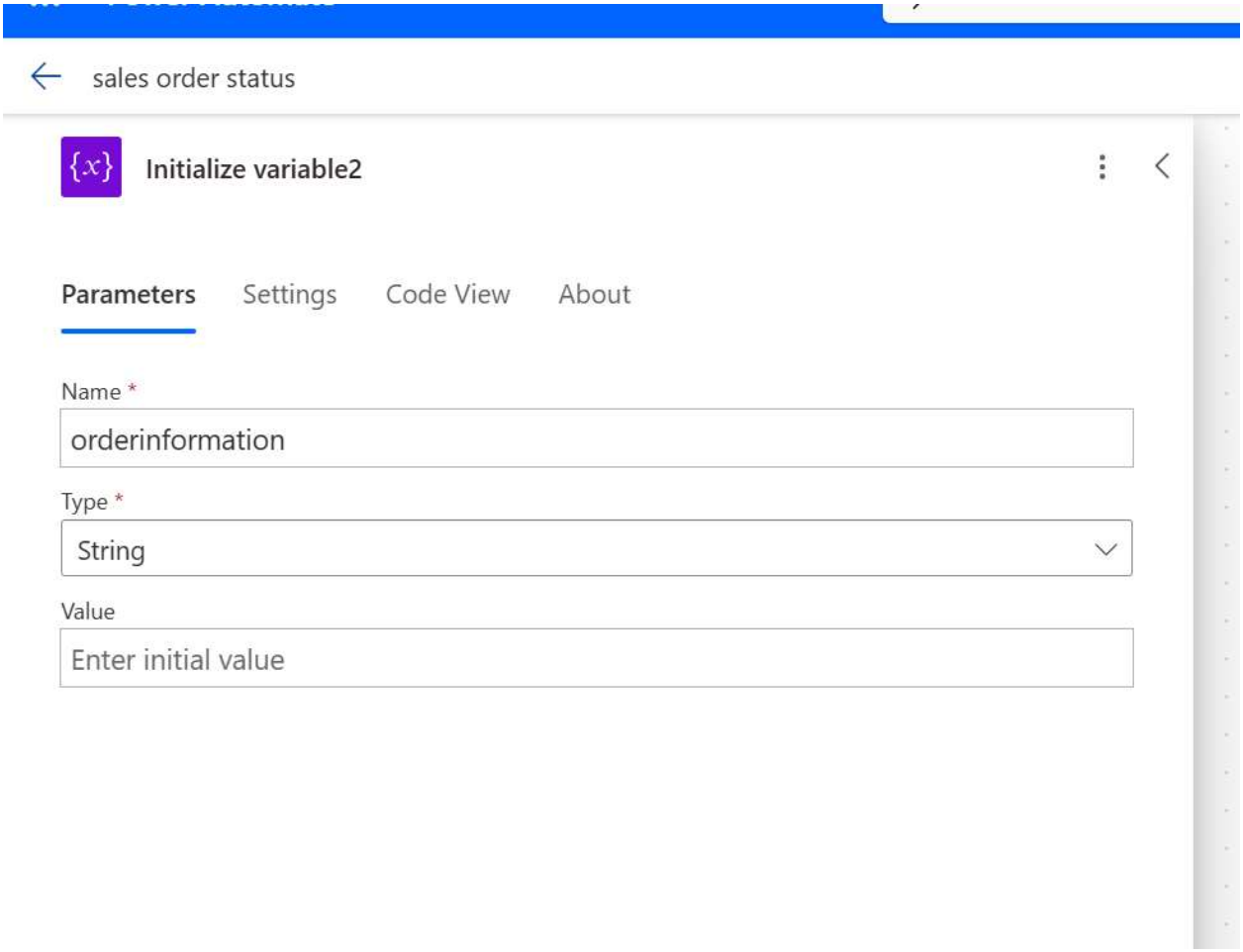
Next, we initialize three variables.

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



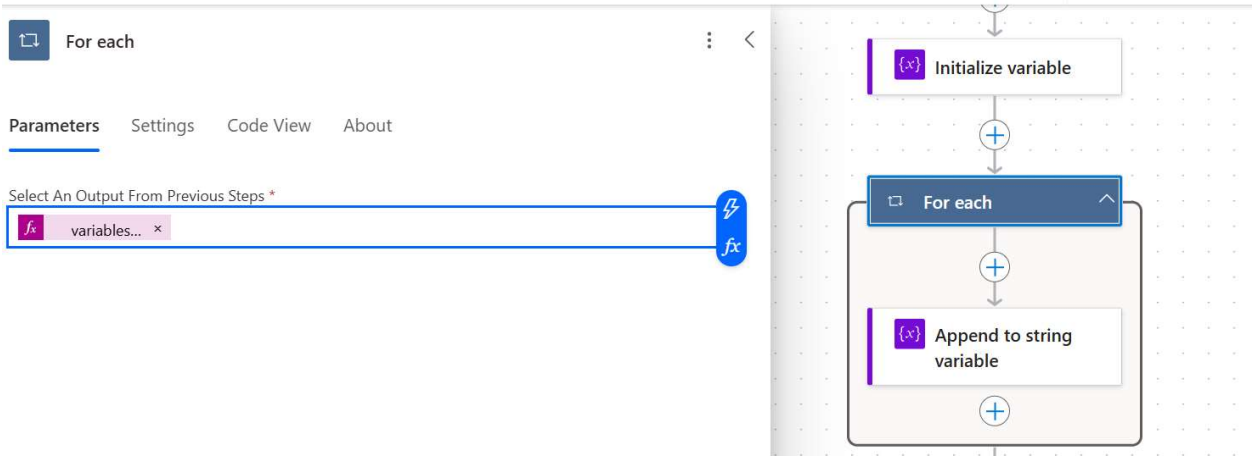


"deliverystatus" of type string which will store the delivery status of the order.

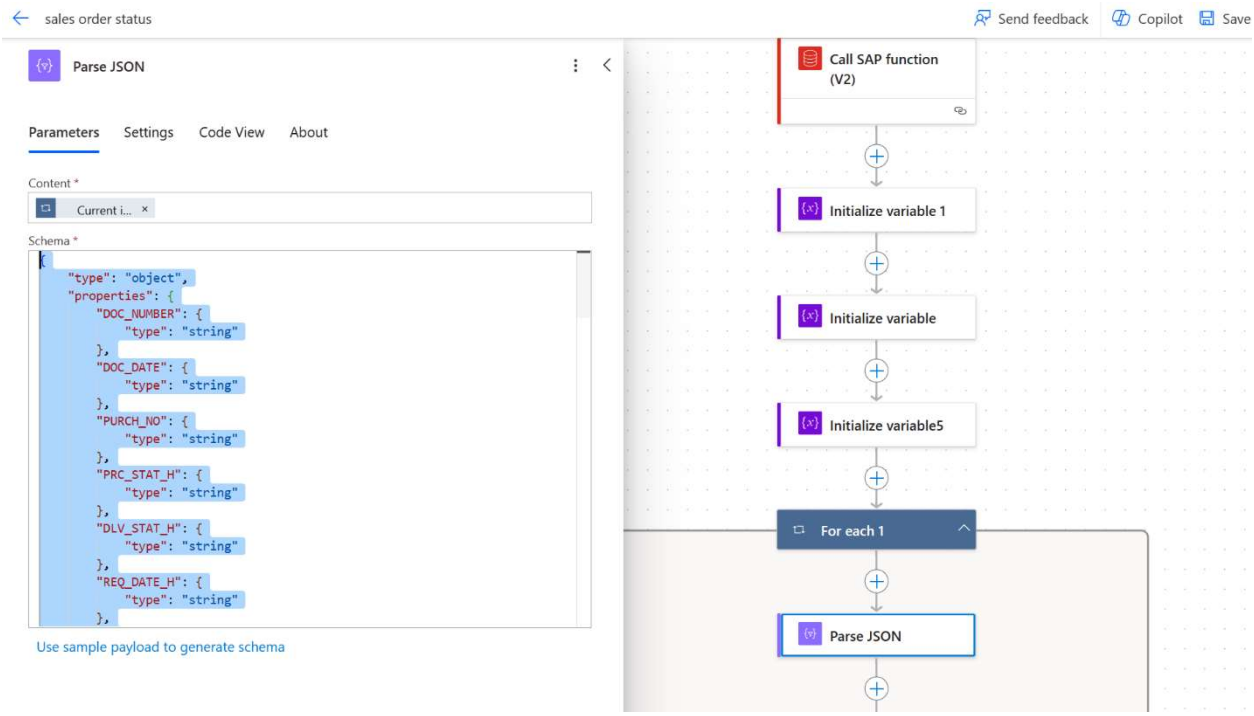


And the variable "orderinformation" which will store other information about the order.

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.



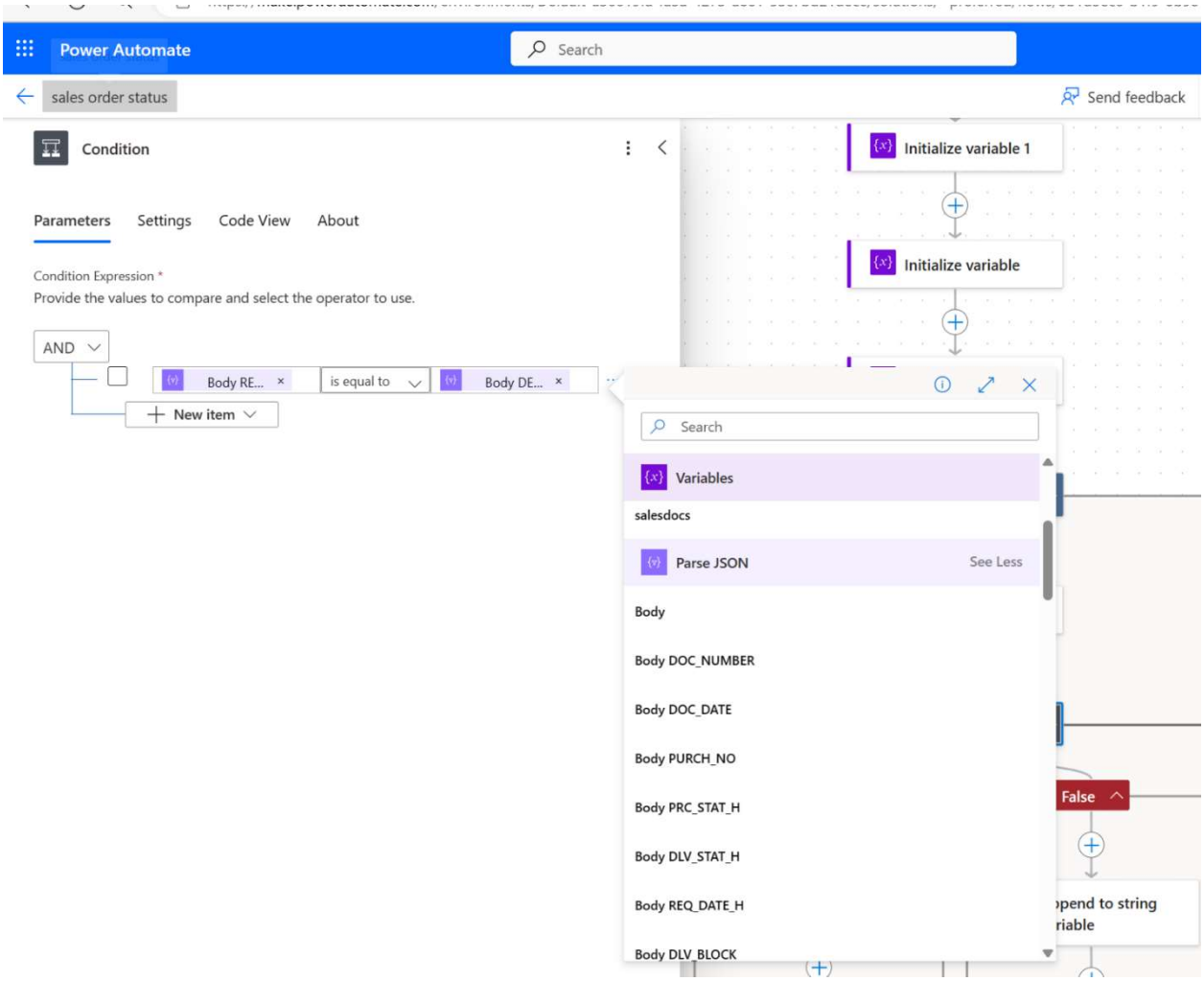
Next, like the previous Power Automate flow, we must parse the JSON response.



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.

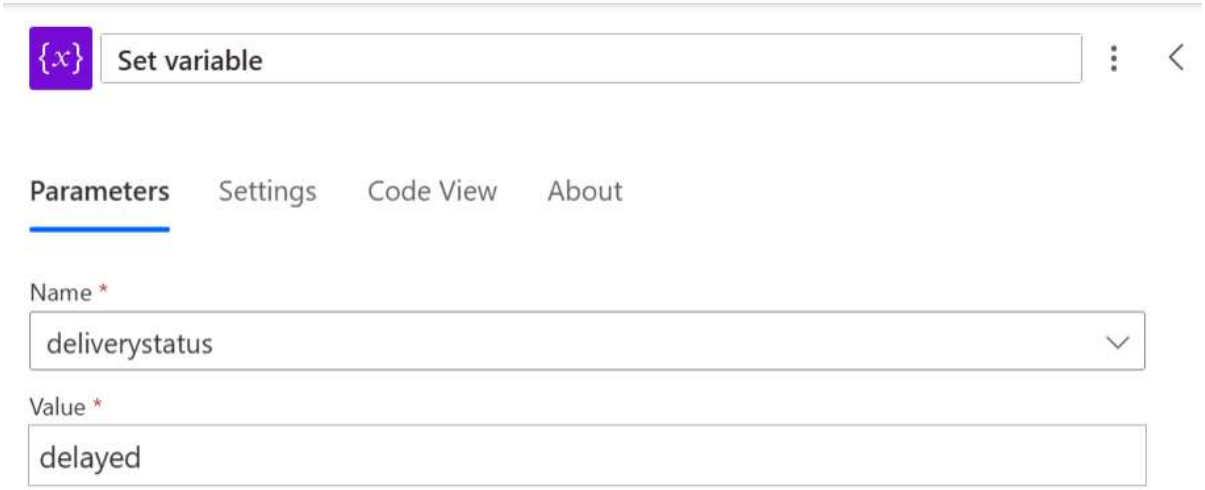
To include this logic in the Power Automate flow, we will add a condition that the field Body.REQ_DATE is EQUAL to DELIV_DATE.

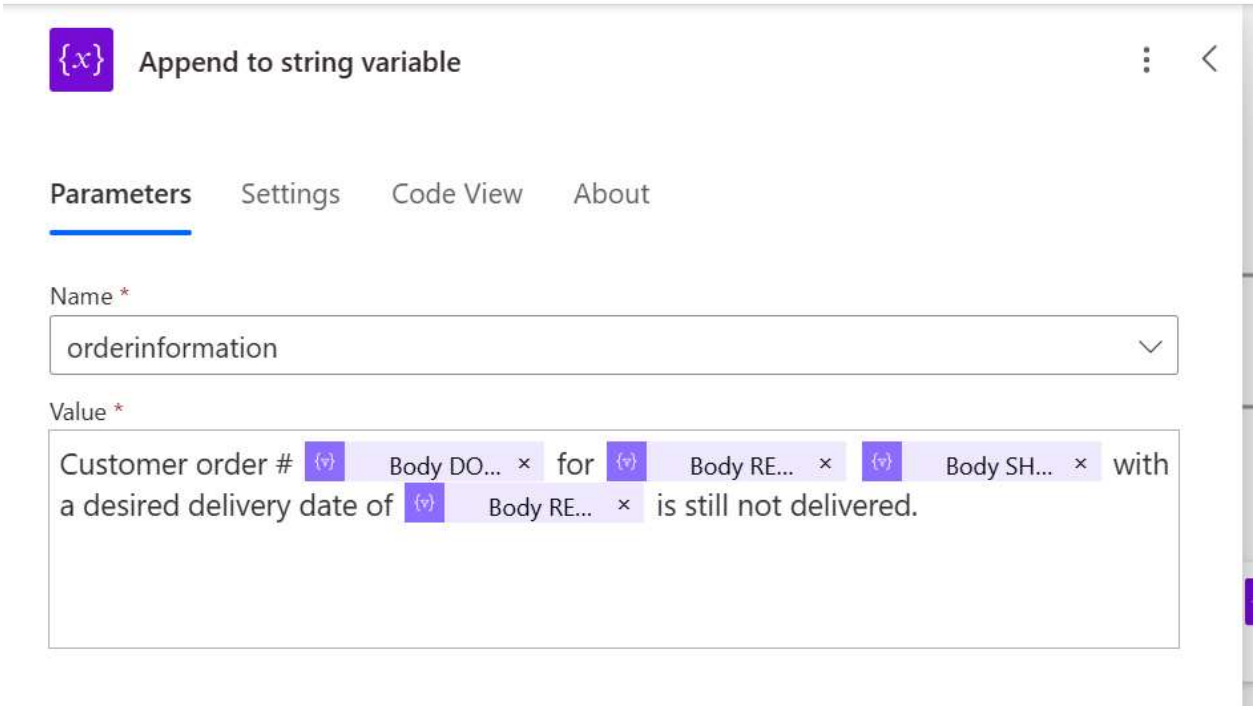


If this condition is true:

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

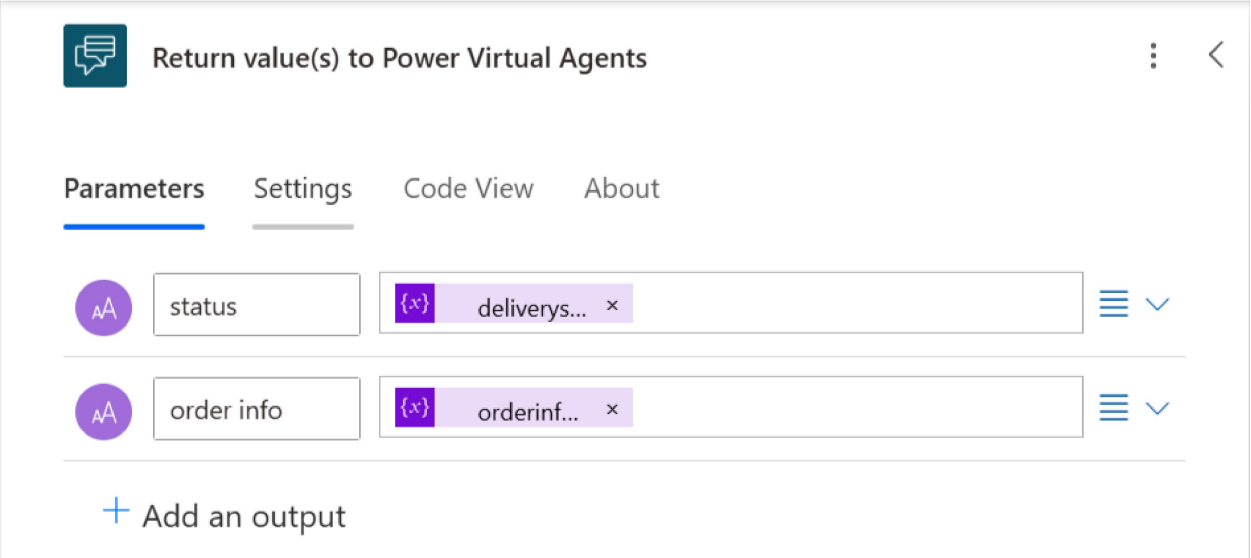
If the condition is not true and the order is delayed, the variable “deliverystatus” will be set to “delayed” the text as shown below will be appended to the variable “orderinformation”.



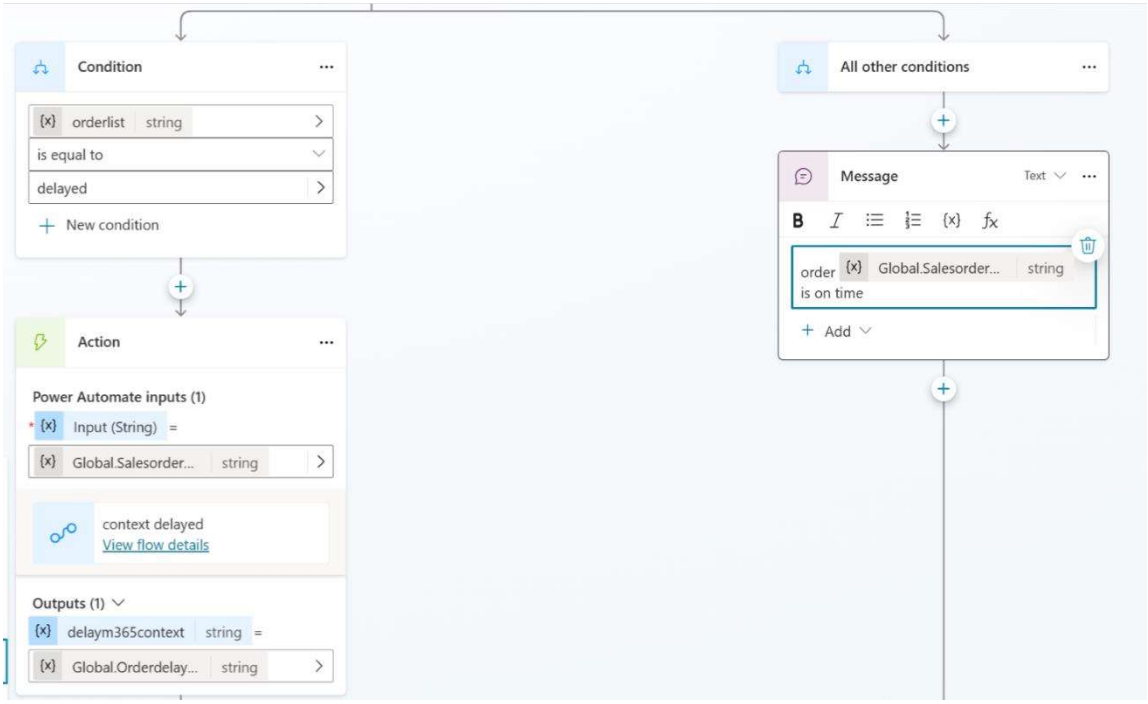


The values can be found in the flow once you download and export it from Github.

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



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

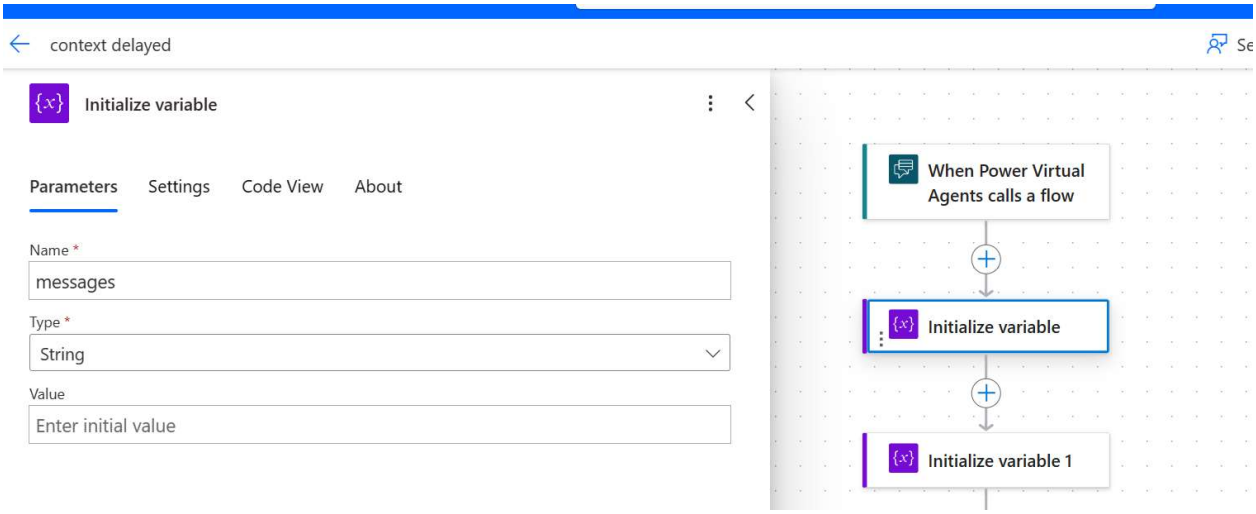


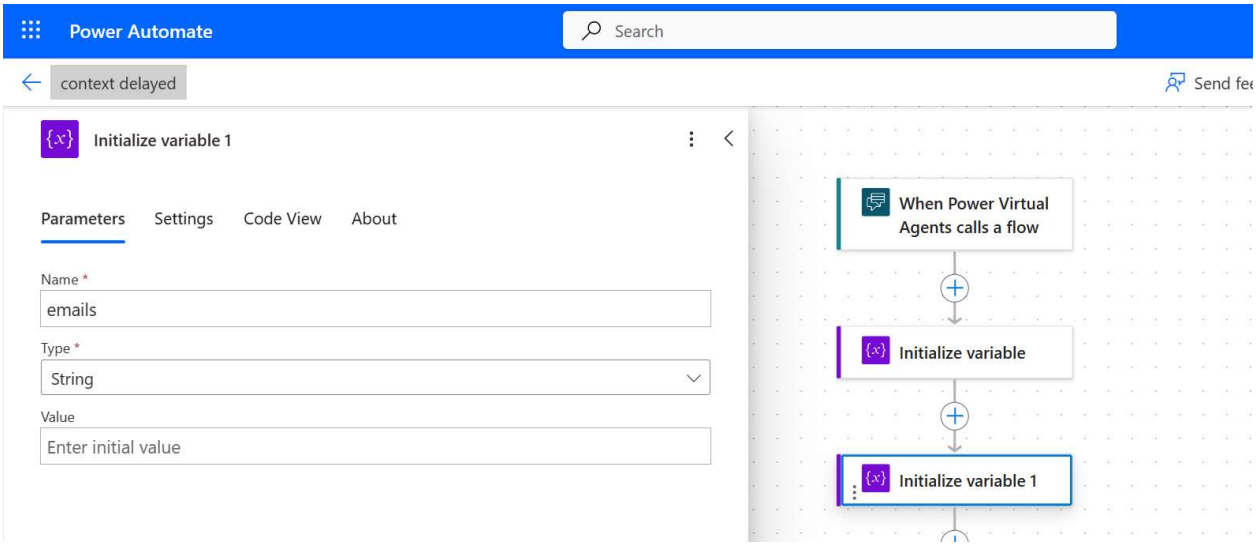
If the order is delayed, the next step is to harness the power of AI and the M365 data for conversation context (emails and Teams' chats on why the order is late). This is done by querying the M365 graph for Outlook emails and Teams' chats related to the order and then passing those emails and messages 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":

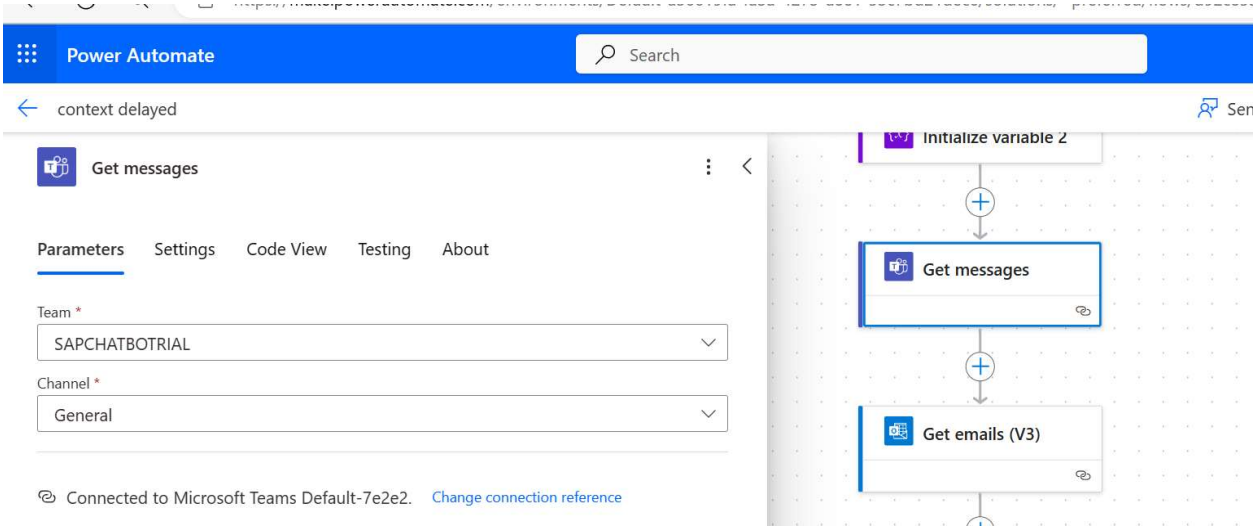
The input for this flow will be the sales order number.

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

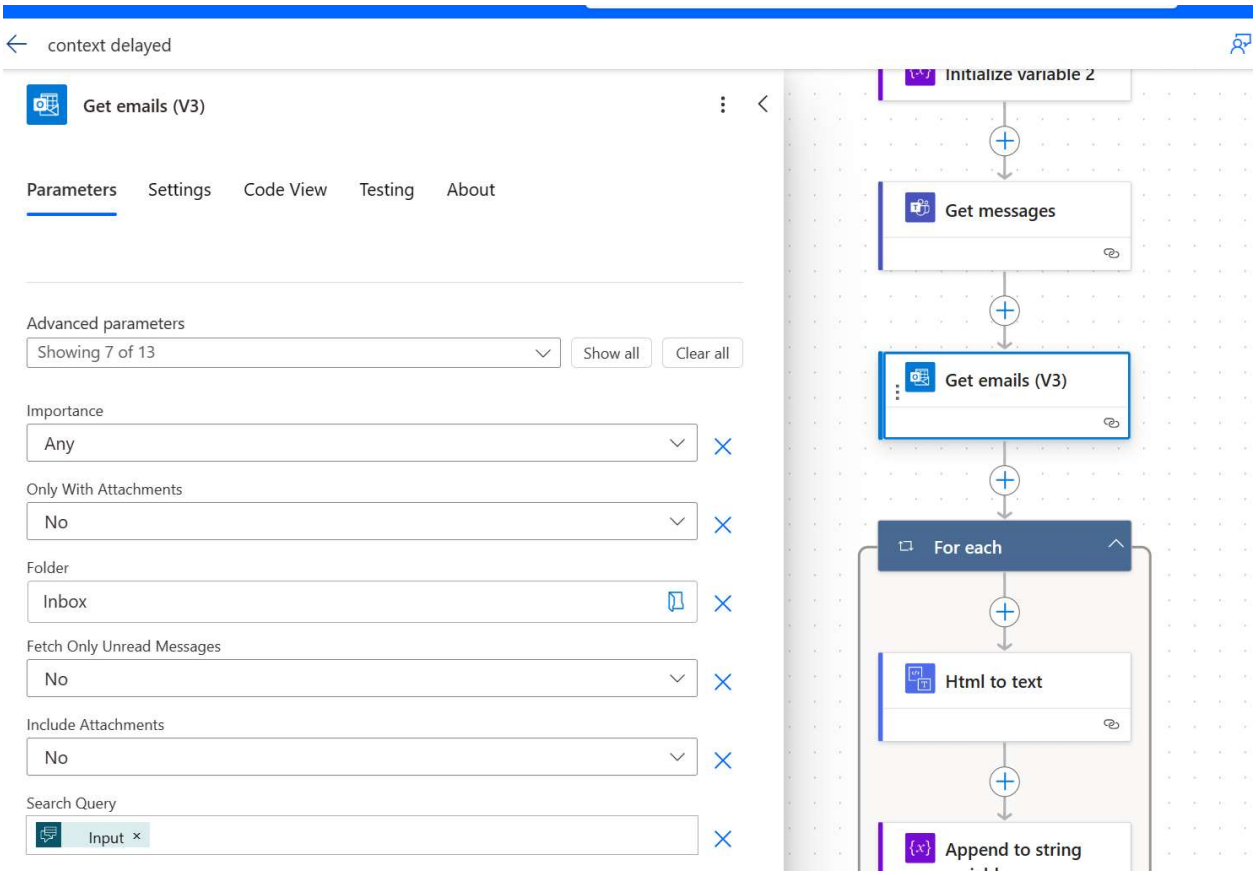




Next, we will also make use of the Team's connector as shown below, however th,e Teams connector does not yet allow you to query specific messages, therefore there will be a bit more processing on that front as shown below, for now we will just fetch all the messages.



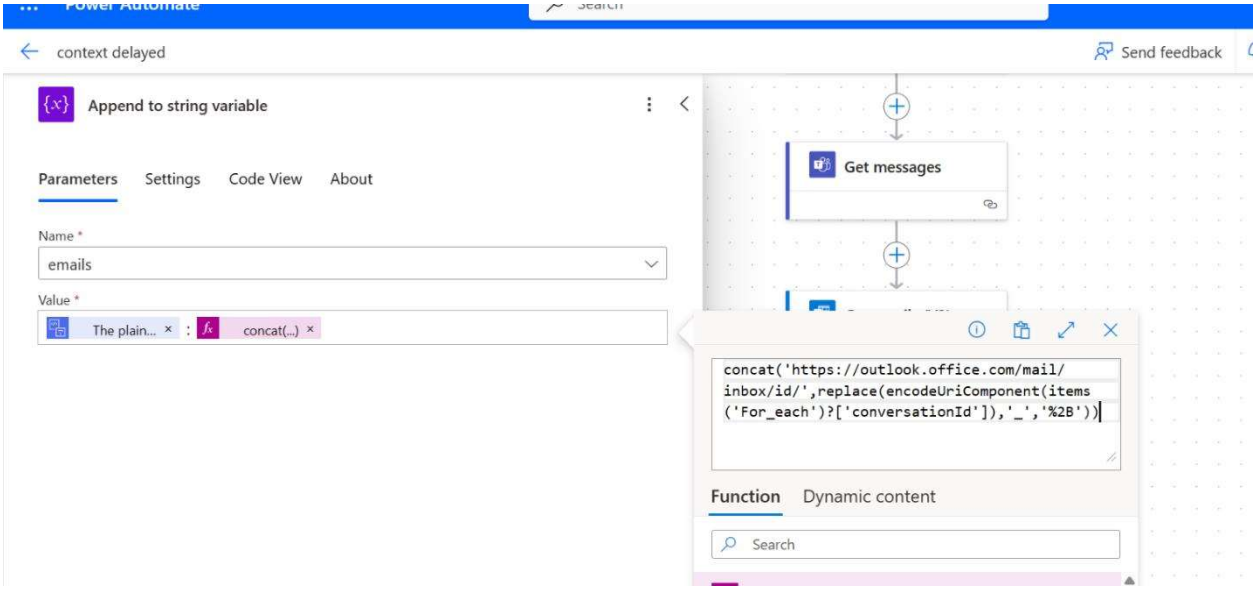
Next, we will fetch all the outlook emails related to the order using the input order number as the search query .



The 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.

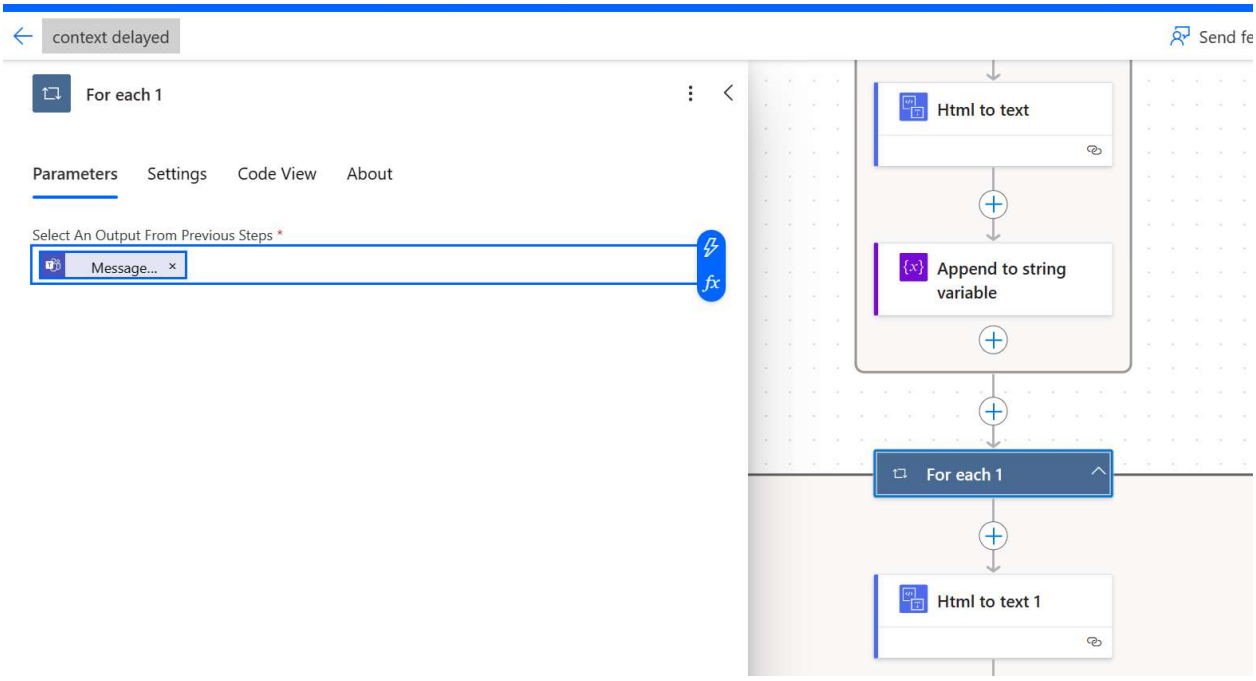
After we get the plain text, we will append it to the email's variable along with the URL of the email, follow the format of the URL as bellow or refer to the exported flow on GitHub.



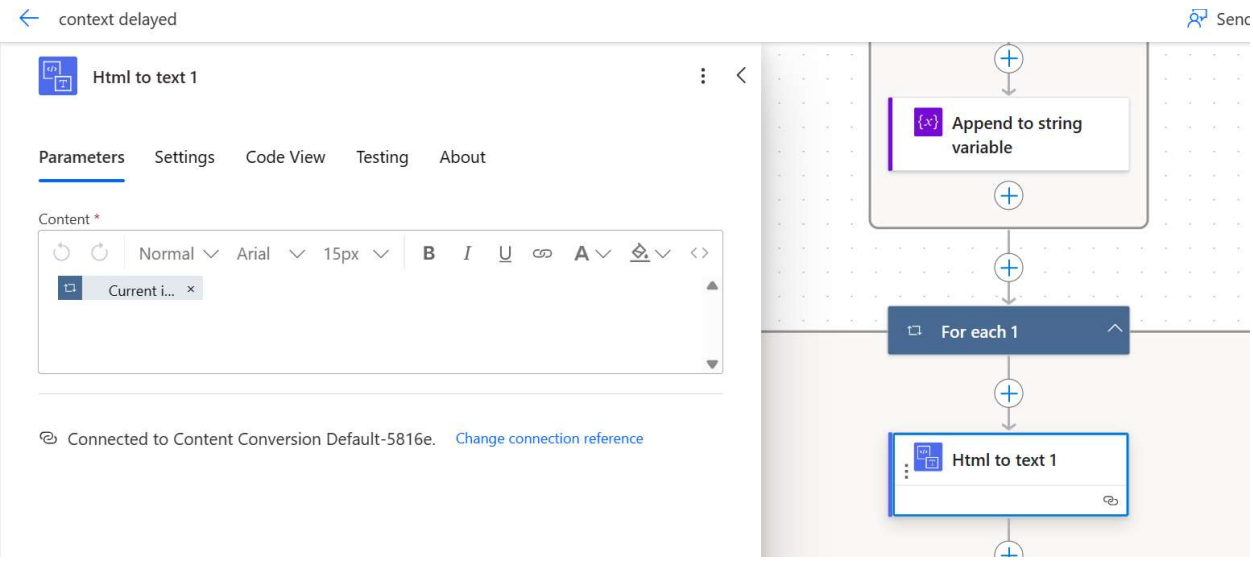
Once we have gotten the emails, we will now work on getting the Teams' messages.

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.

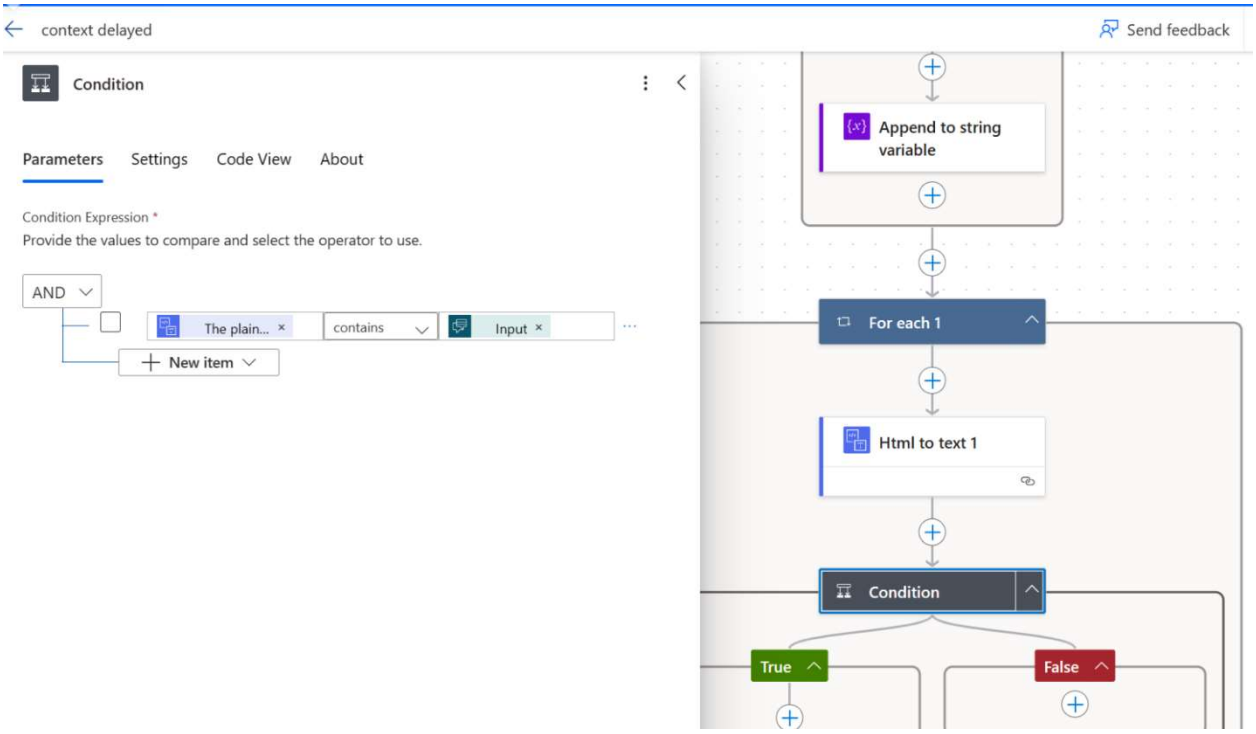
To do this we will loop over all Teams' messages:



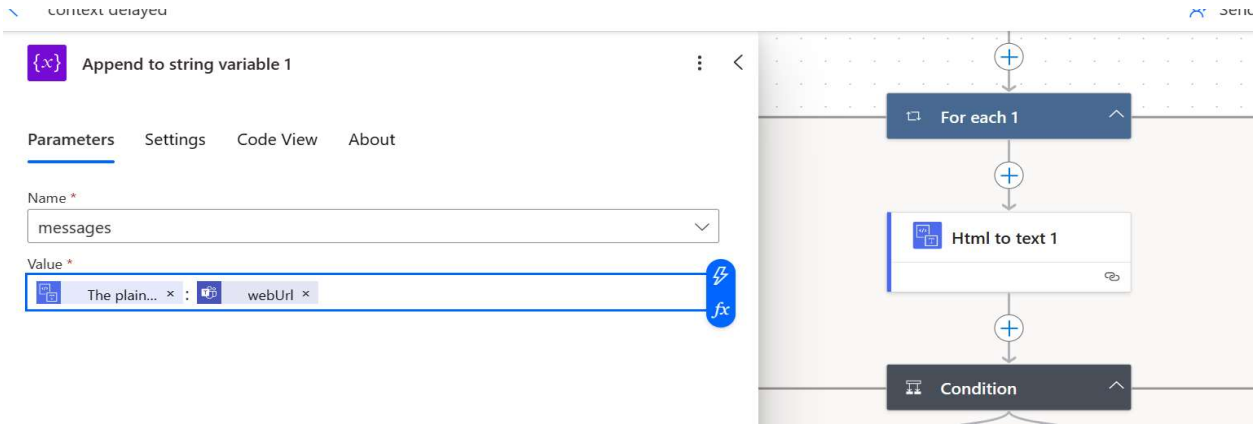
Next we will convert the response from the HTML format to text using the “Html to text” action.



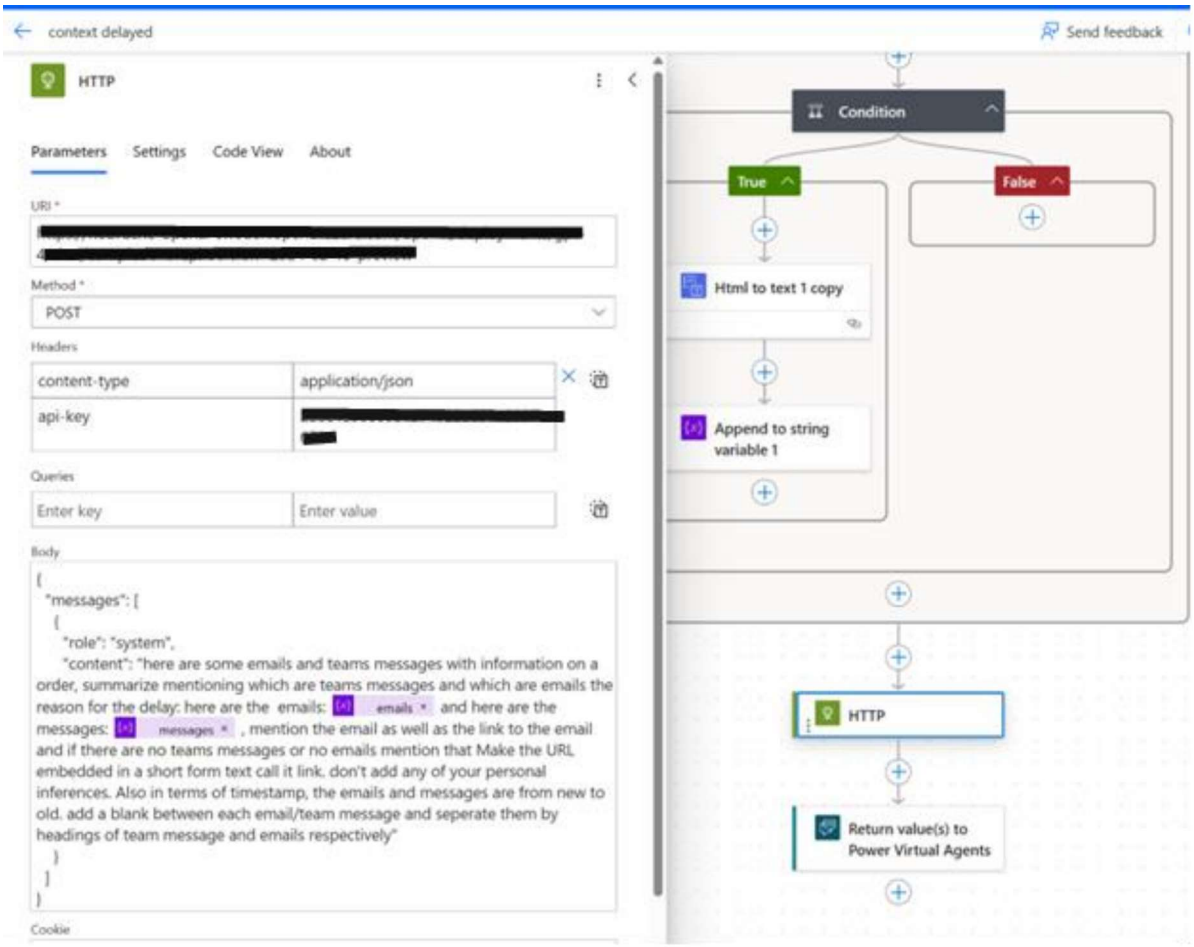
Once we have the Teams message in plain text, we will use the “Condition” action and see if the content contains the order number (gotten from the input variable).



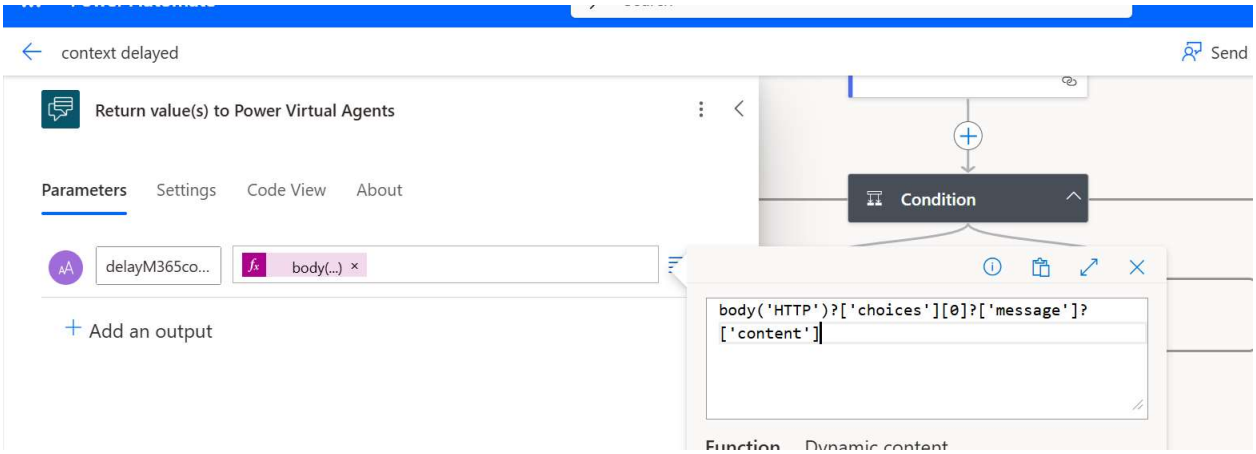
If it does, we will append the message as well as the WebUrl of the Teams message (gotten from the dynamic variables)



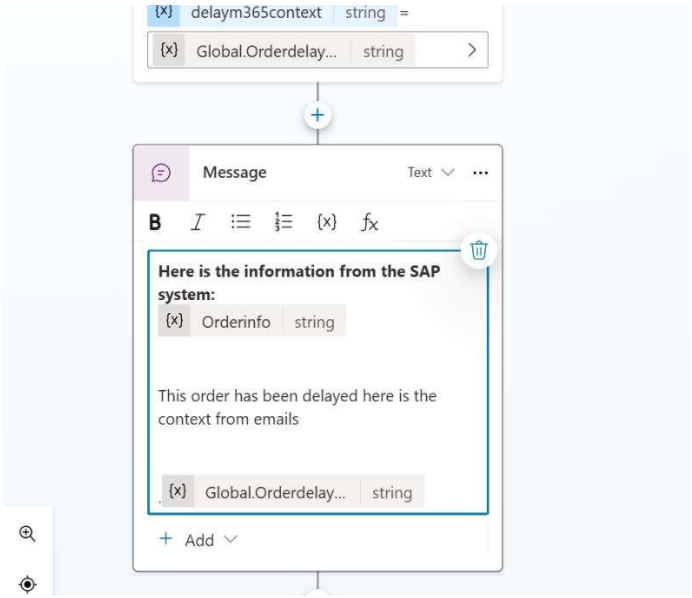
After the relevant Teams' messages and the URLs are saved in the variables, we will now send this as an input to Azure OpenAI to summarize and give the salesperson insights on why the order is late. For this we will make a HTTP request to Azure OpenAI as shown below.



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

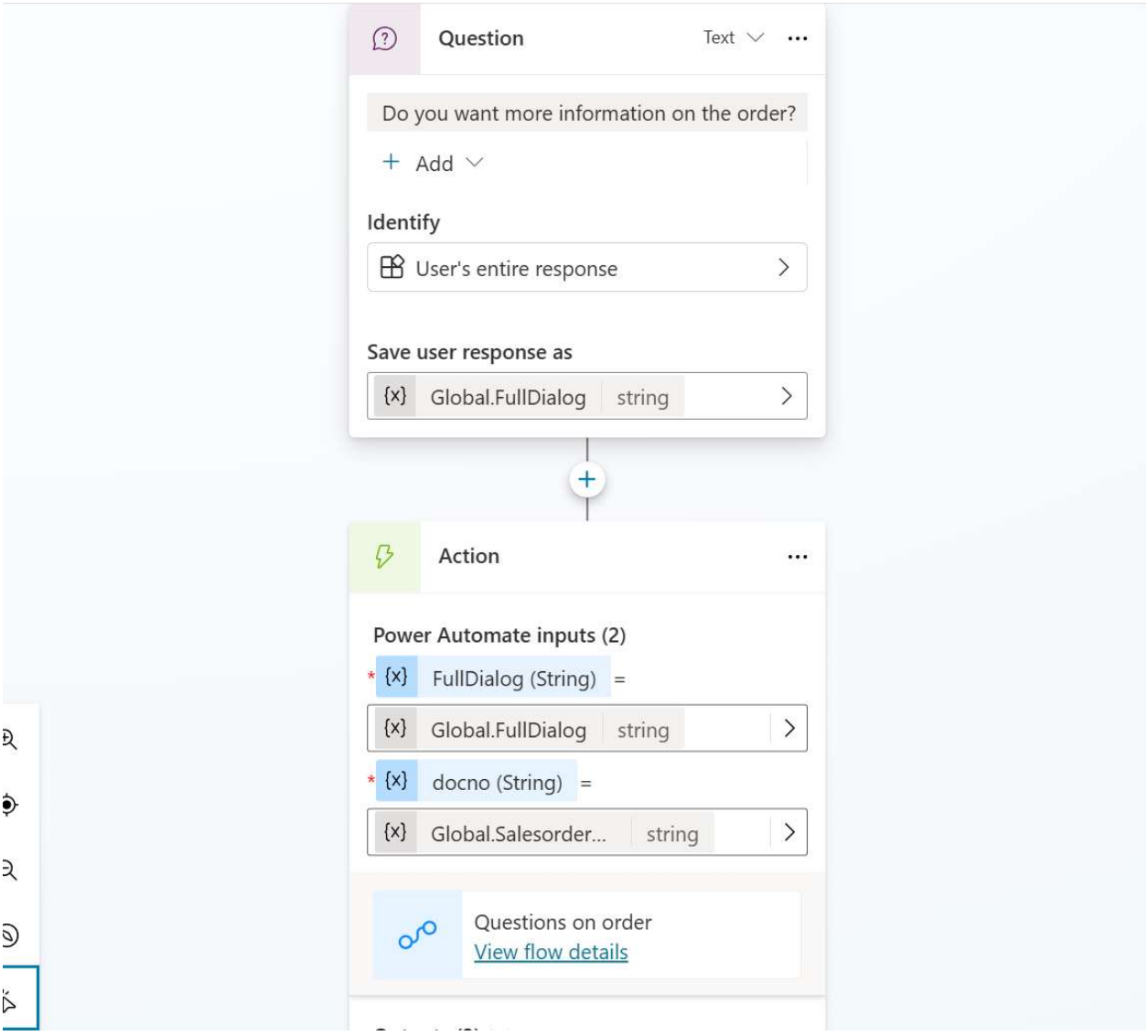


Finally, we will send the response back to Copilot Studio and present this information to the user as shown – the variable "orderinfo" is the output from the "get order status" step.



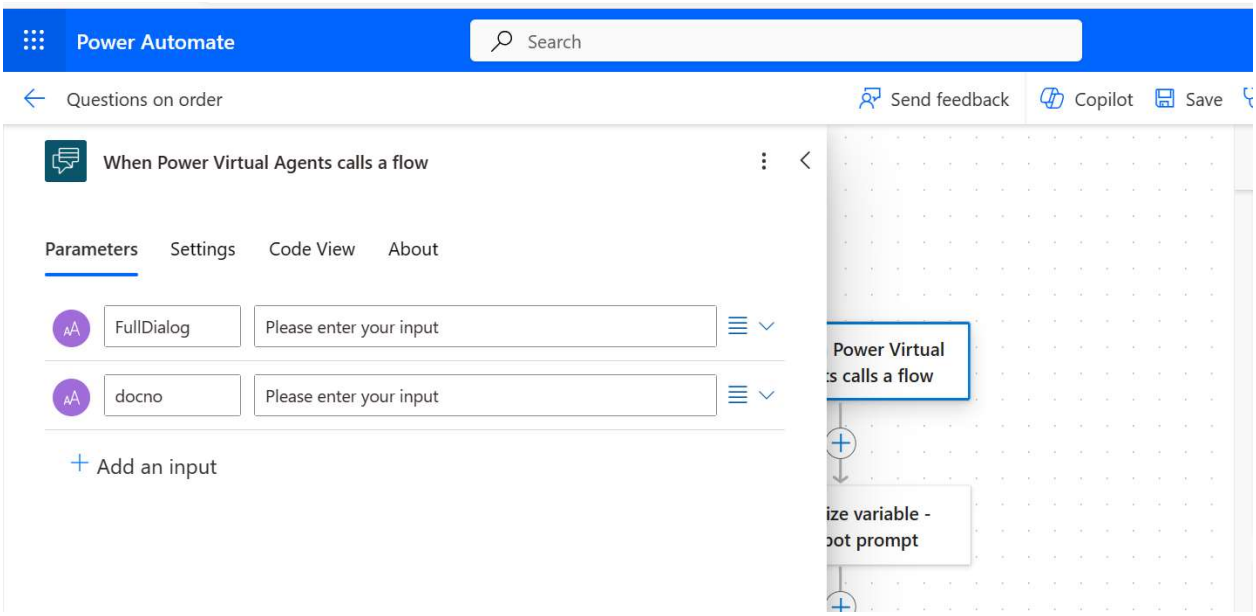
The next step after giving the order context is to allow the user to ask the bot any question they have about the order on the SAP data from the system.

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.

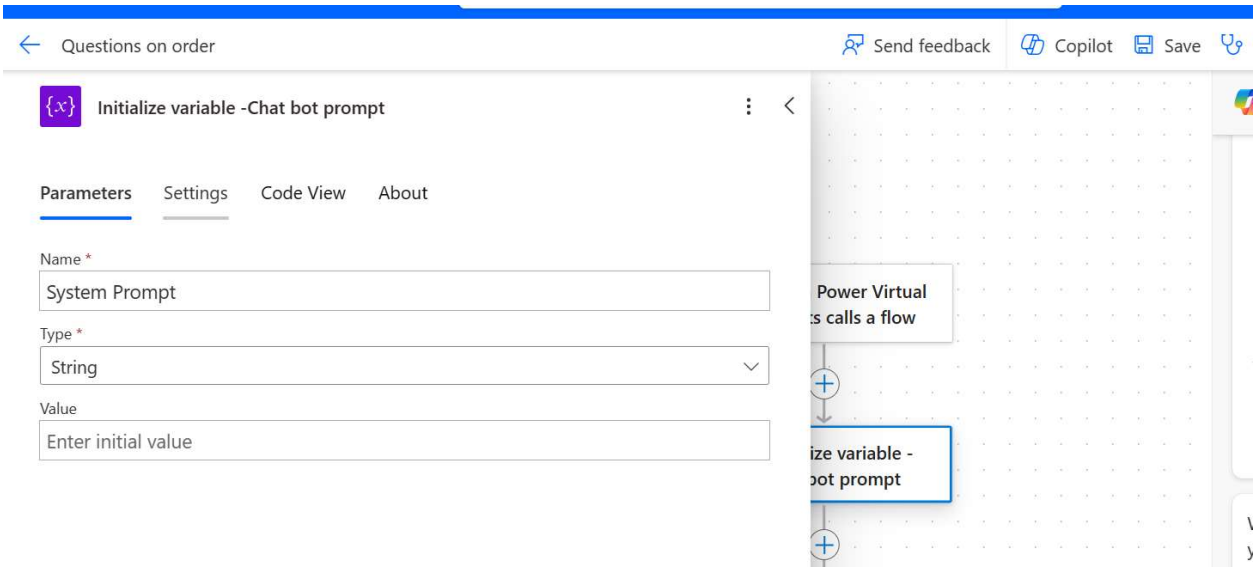


Here is the Power Automate flow for this, the name of this from is “Questions on order”:

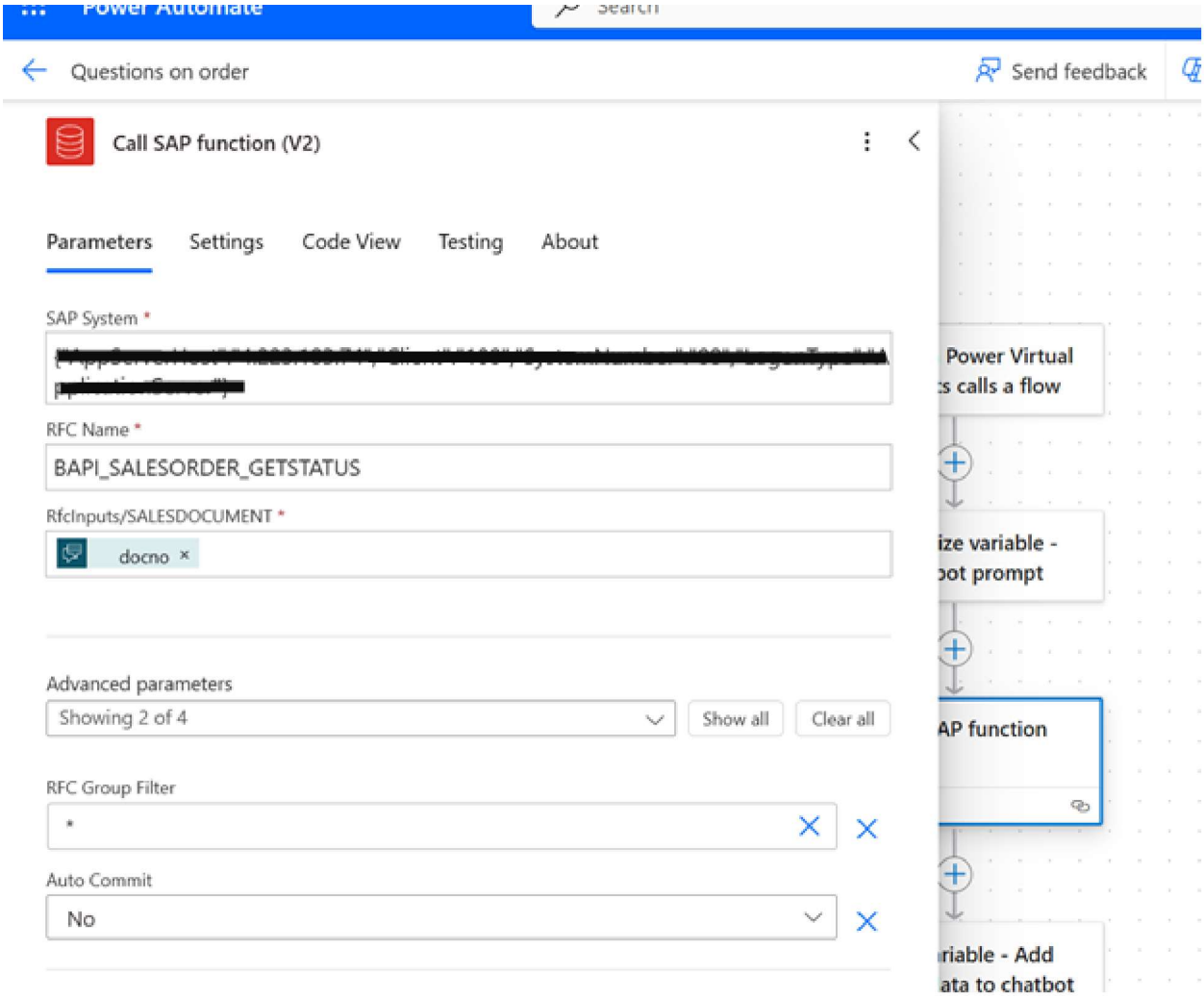
The first step is to add two inputs to this, the document number and the question that the user is asking.

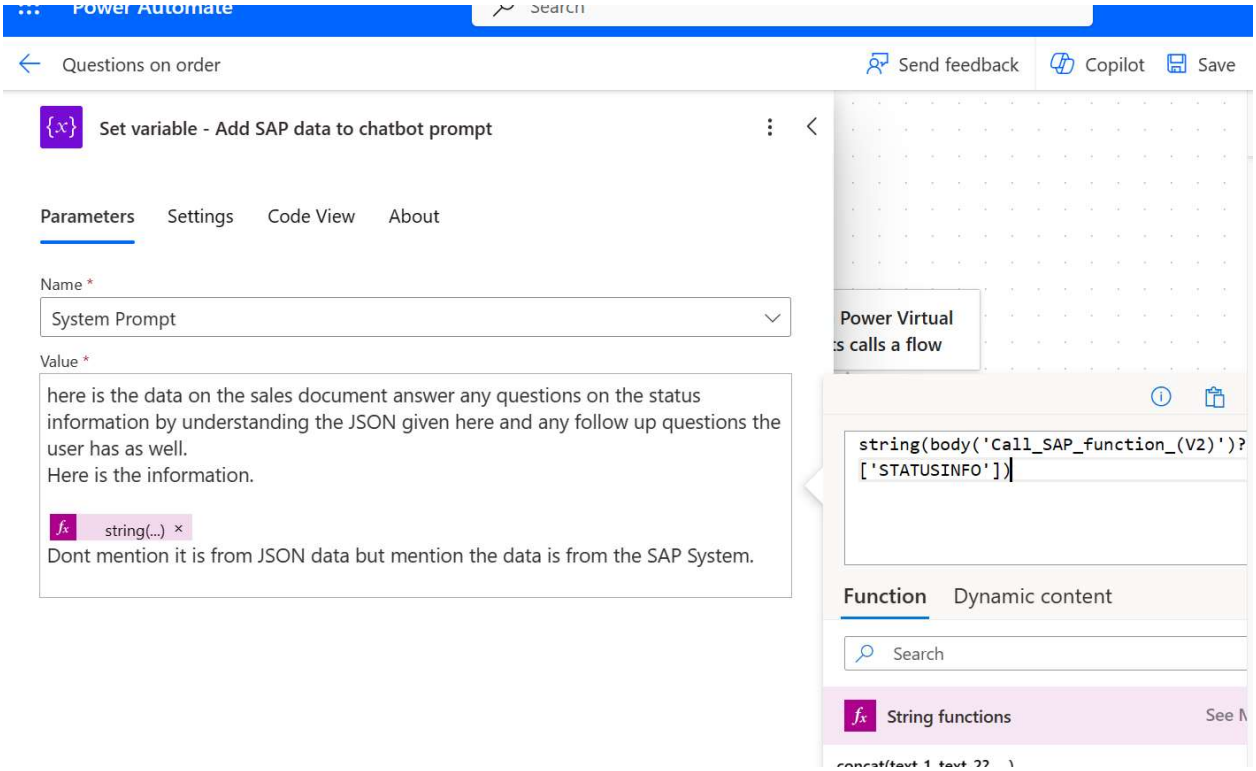


The next step is to initialize the prompt variable that we will send to Azure OpenAI.

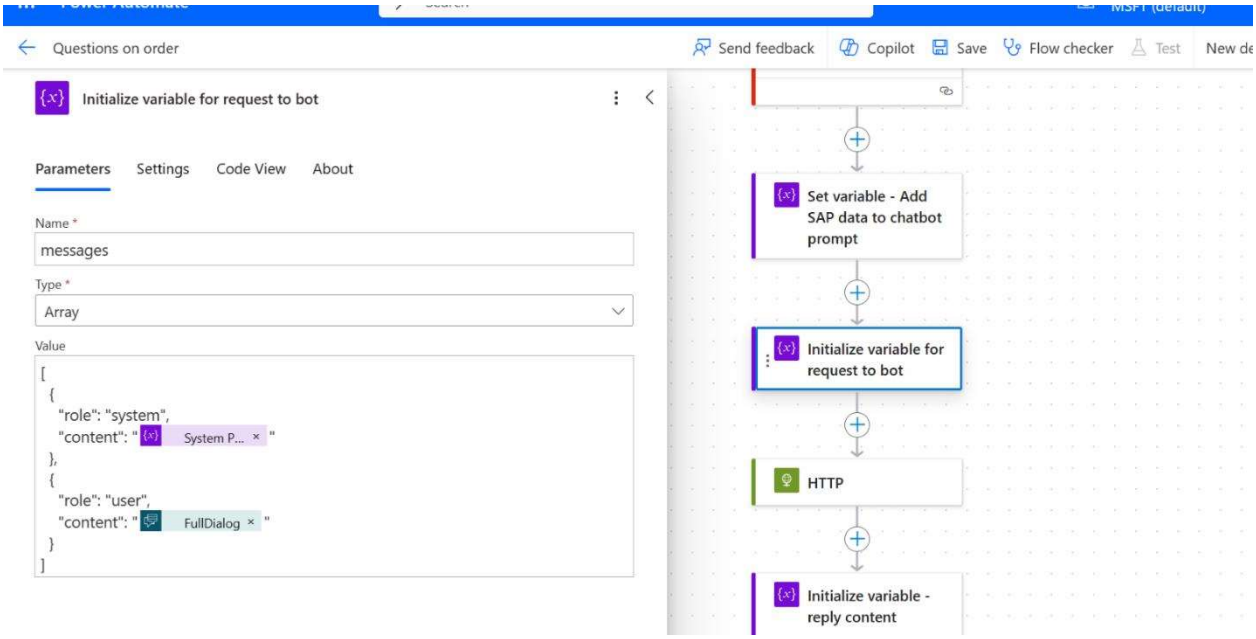


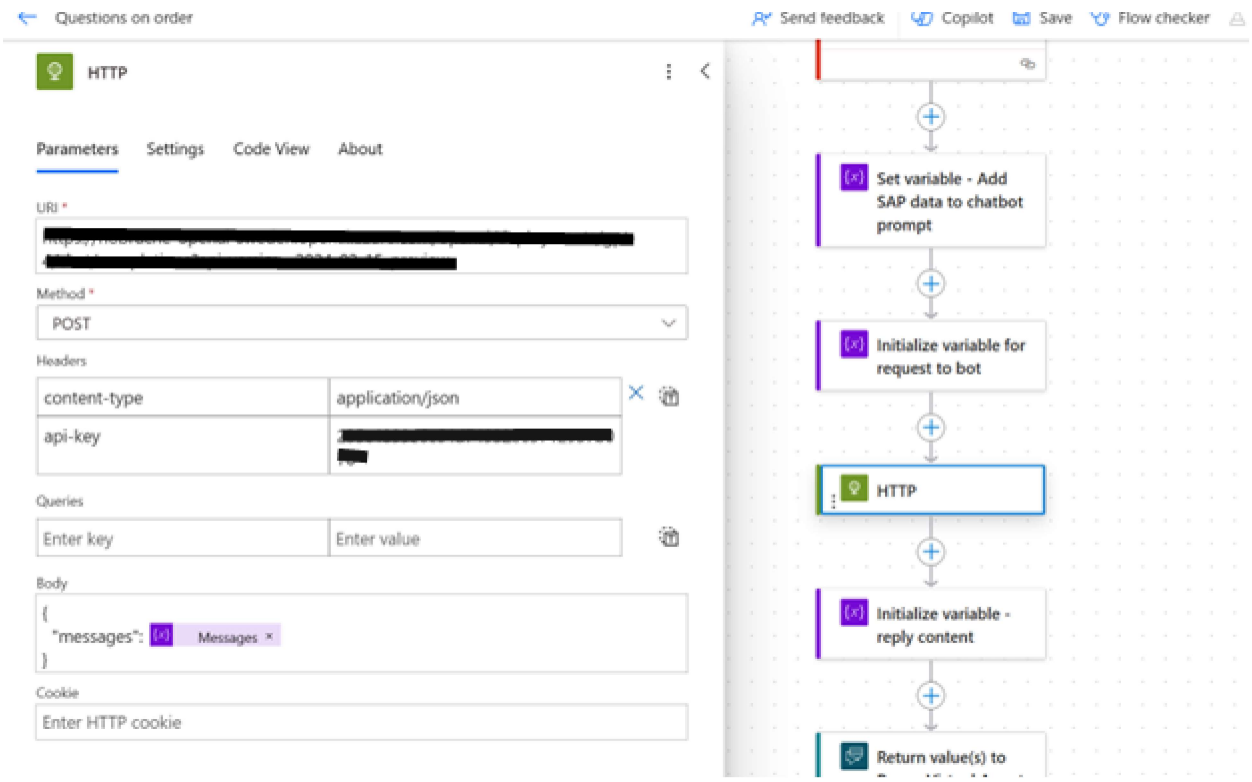
The next step is to call the SAP function to get order status.



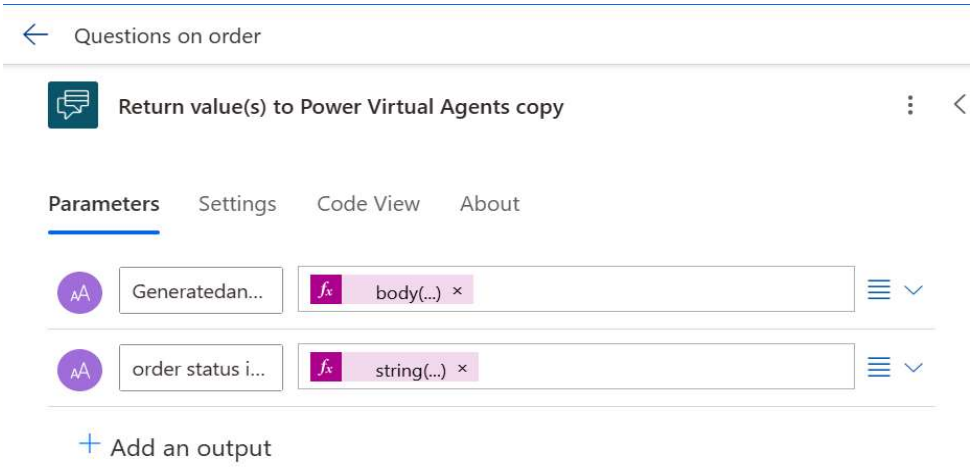


Next, we will add the user's question and the SAP JSON response to the HTTP Request to Azure OpenAI.

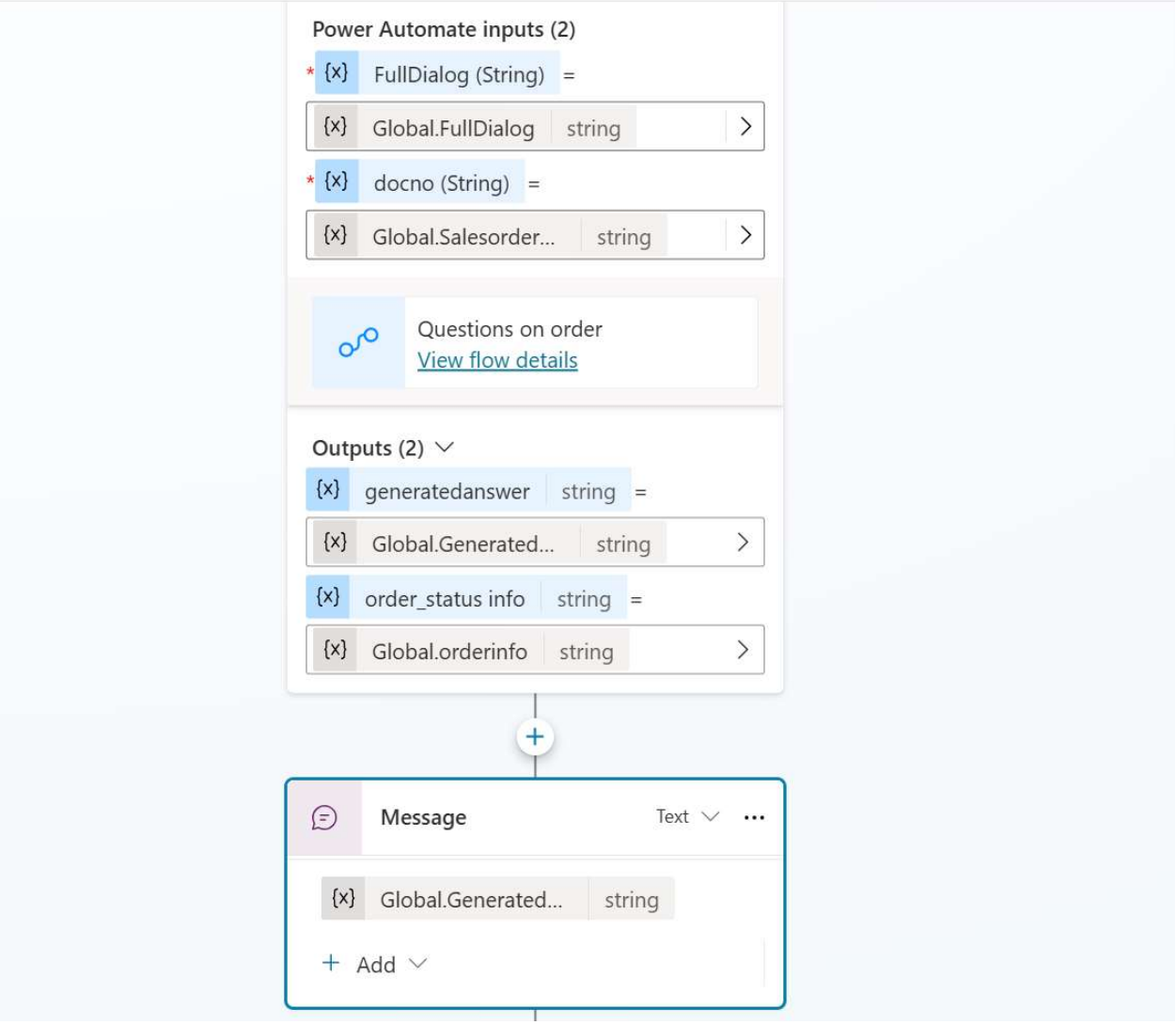




Lastly, we will process the response, store it in a variable, and send it back to the Copilot Studio – this variable will be called "Generatedanswer". (Note: the value for the response is the same formula as the previous Power Automate flow: `body('HTTP')?['choices'][0]?['message']?['content']`)

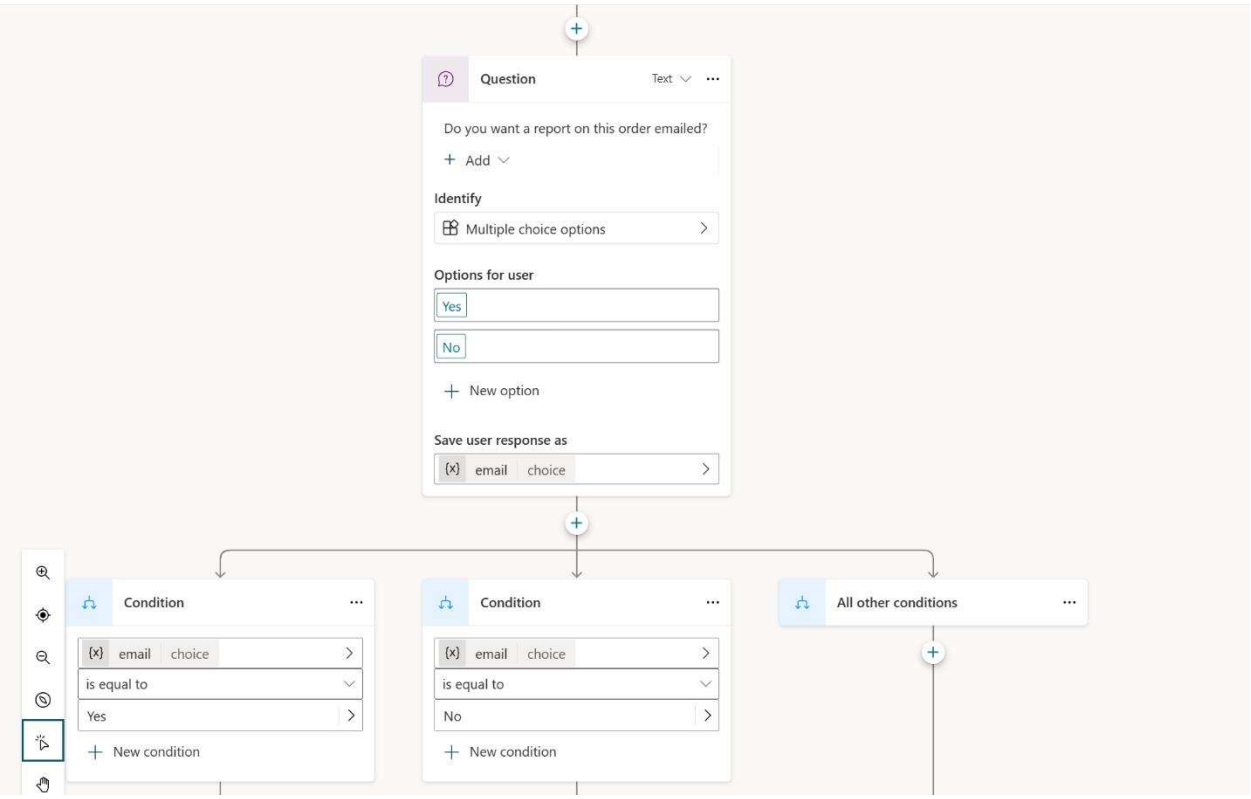


The other variable we will send back is the order status information we got from the SAP System. This variable could be used by other potential topics, and we save it as a global variable in the Copilot Studio so that it can be used across topics.

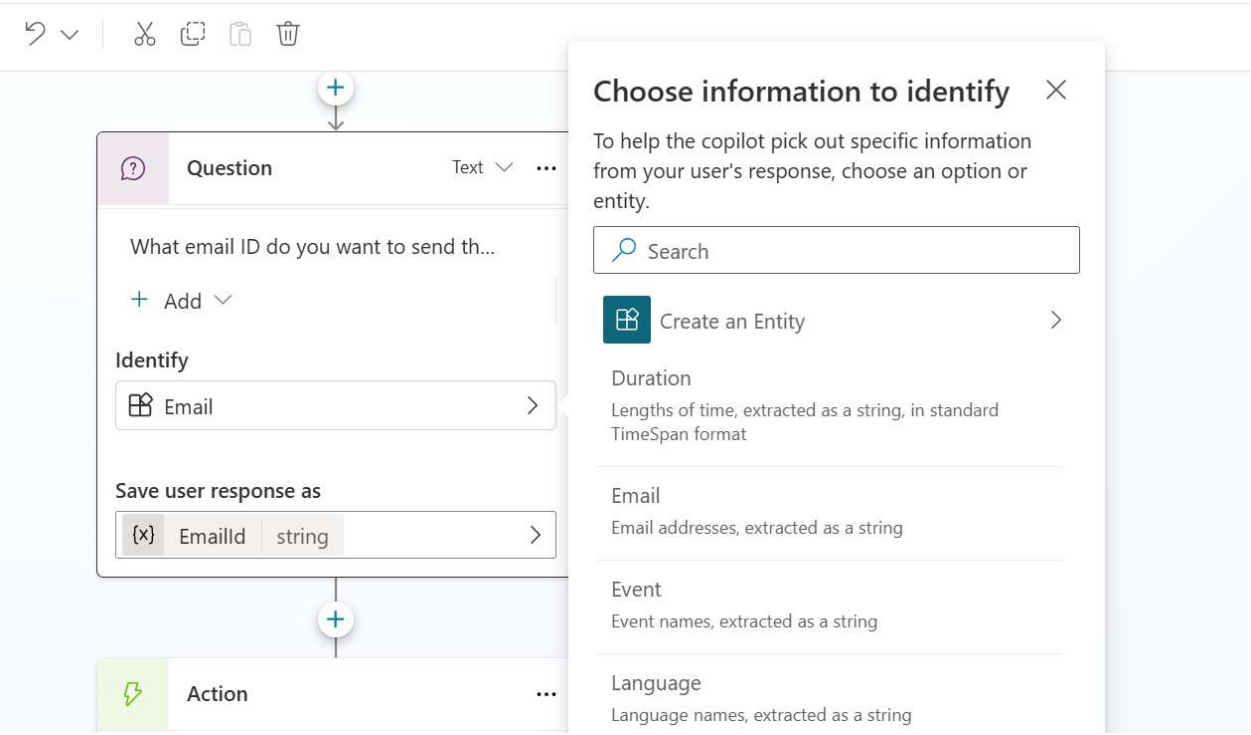


Lastly, if the salesperson wants to get back to the customer with the analysis, we will ask the salesperson if they want the entire conversation/ questions they asked about the order to be summarized by AI and put into a report to be sent to the customer (or anyone via Outlook).

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

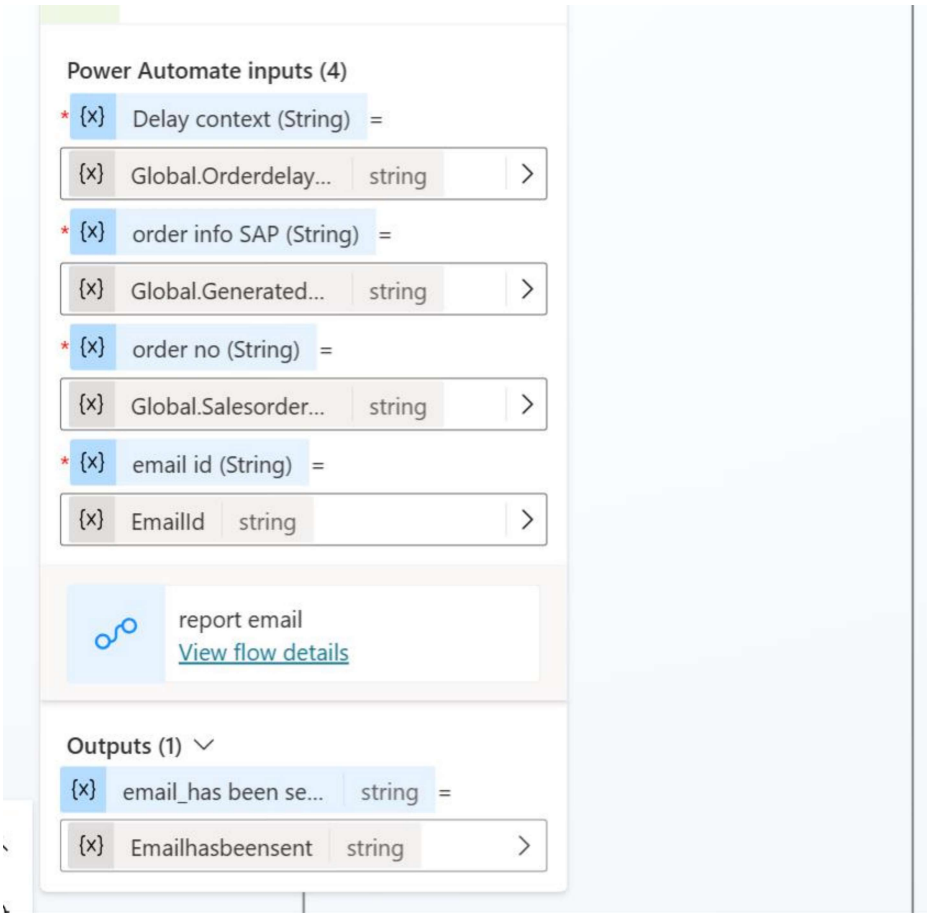


If the user responds that the they want to send an email, we will ask them for the email ID they want to send this to, for this, we need to ask another question and select the readymade email entity as the response (as we did for customer number and sales order number earlier).



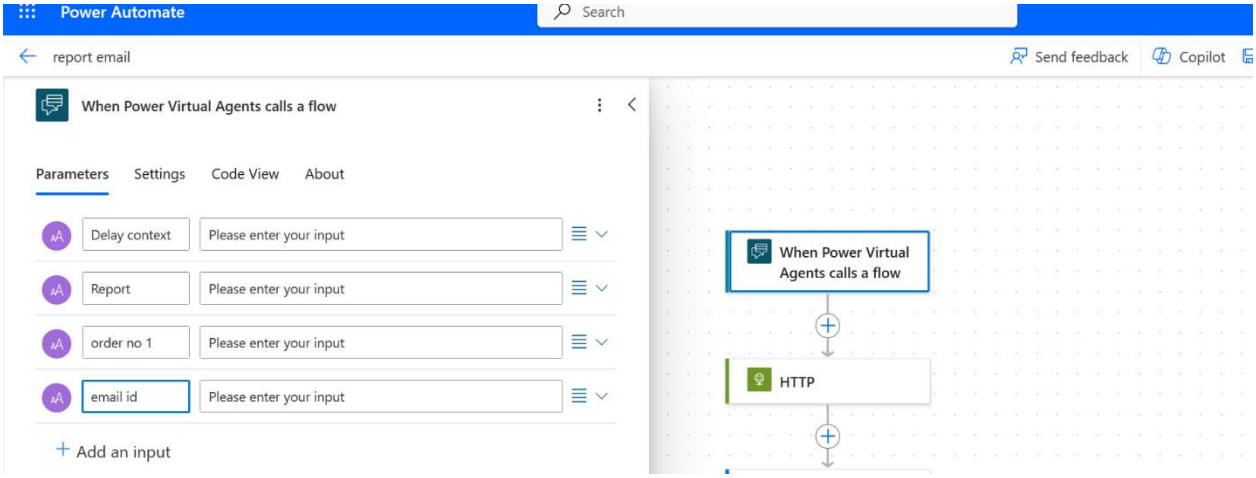
Finally, after that, we will call the final Power Automate flow with the variables shown below. This Power Automate flow is called "report email"

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

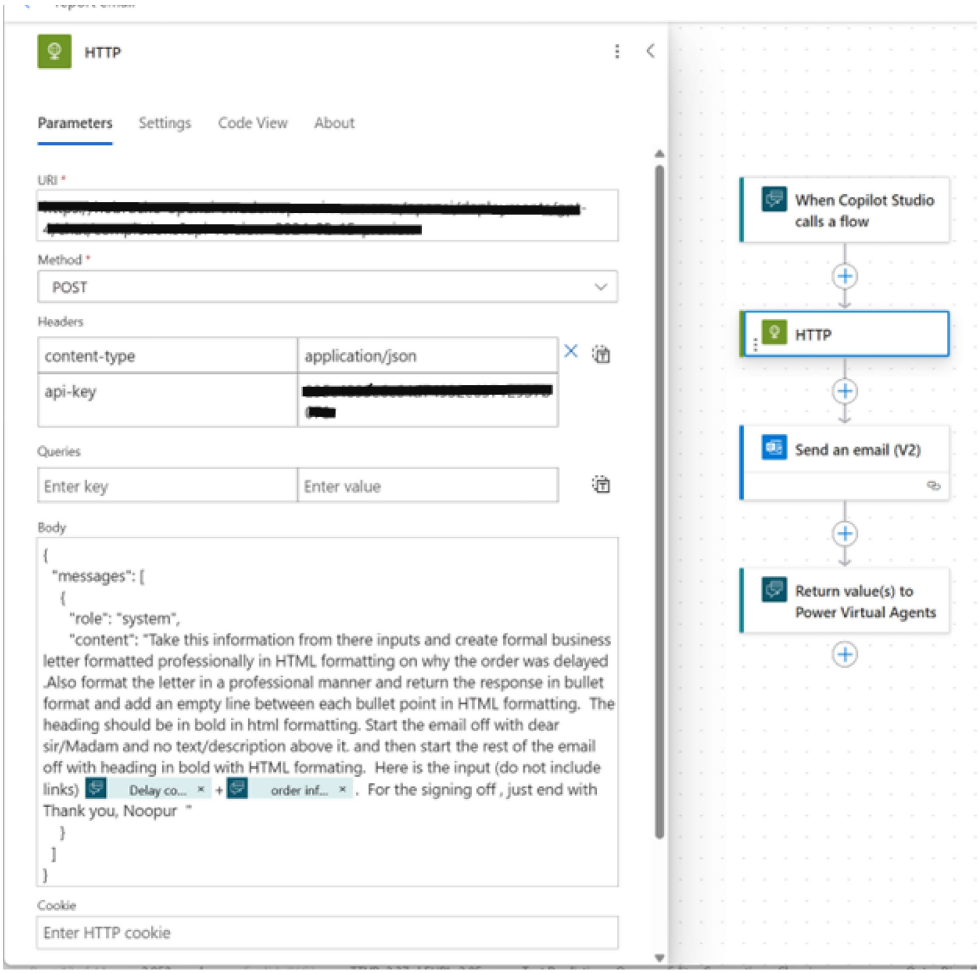


The steps for this flow are:

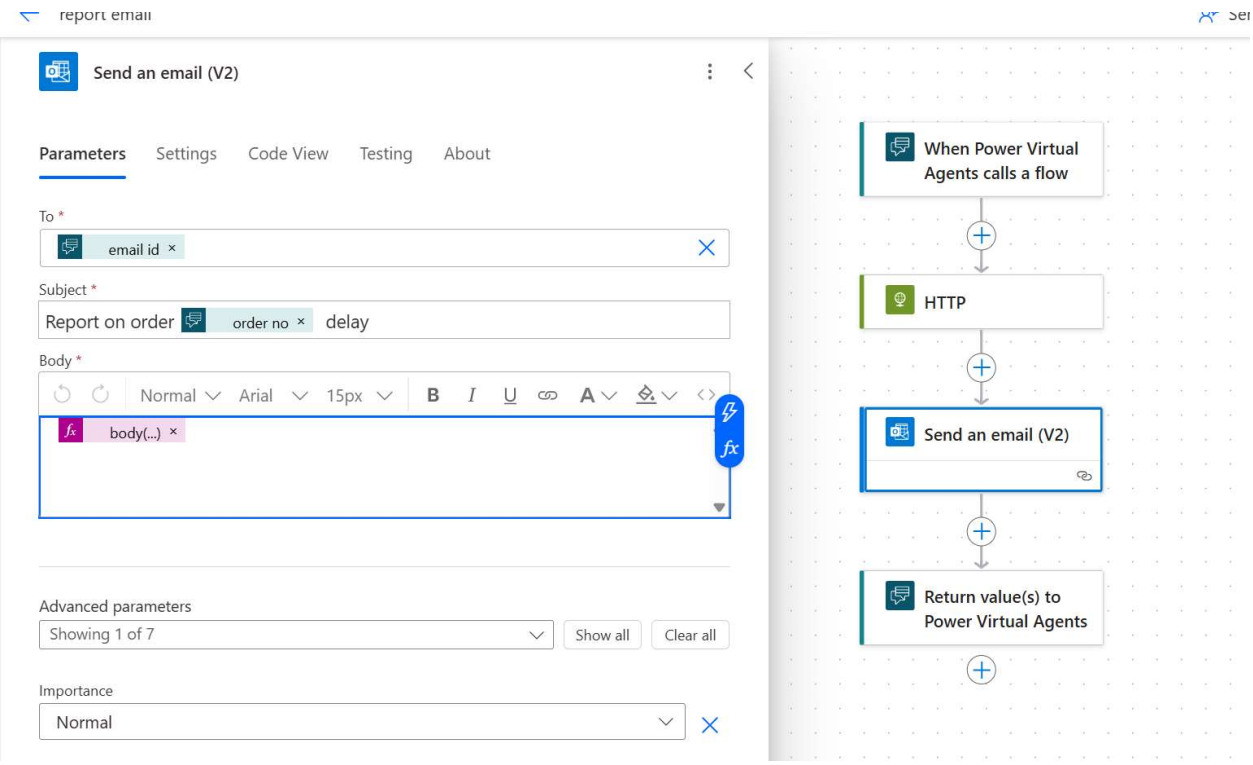
The first step is to define the 4 variables we need. The summary of Outlook + Teams messages, the response to the previous question asked from Azure OpenAI, the order number and finally the email ID we want to send the email to – this would be the customers email ID.



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:

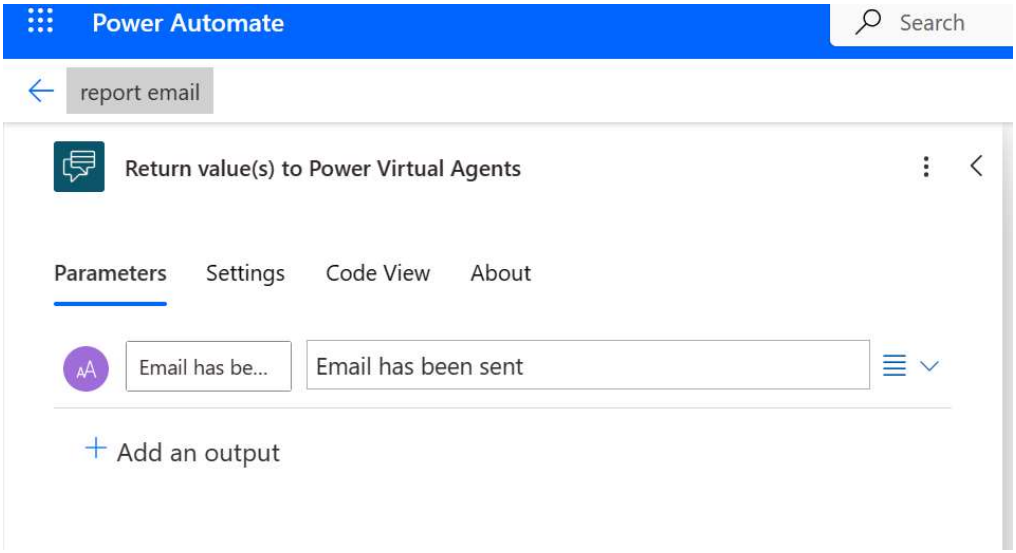


Next we use the outlook send email function to send the email to the customer and populate the fields as shown below.

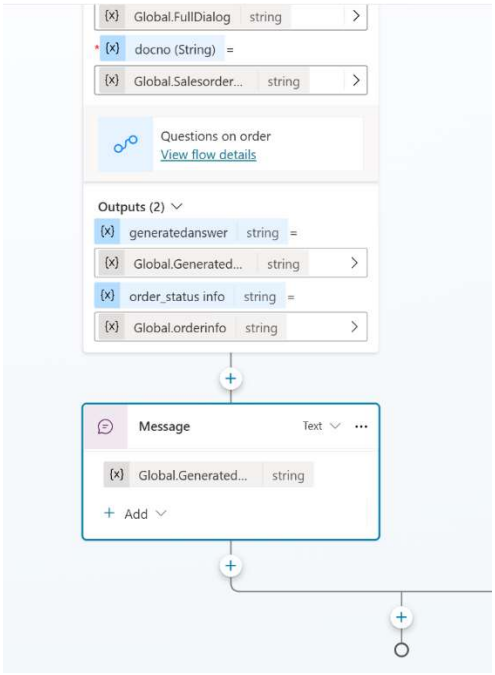


The body of the email here is the response from Azure OpenAI (which is the generated email in this case) - this is the expression for it as used before body “`('HTTP')?['choices'][0]?['message']?['content']`”.

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

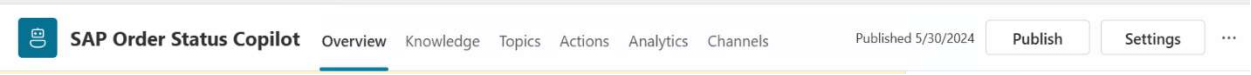


The chat bot sends a final response to the salesperson informing that the email has been sent.

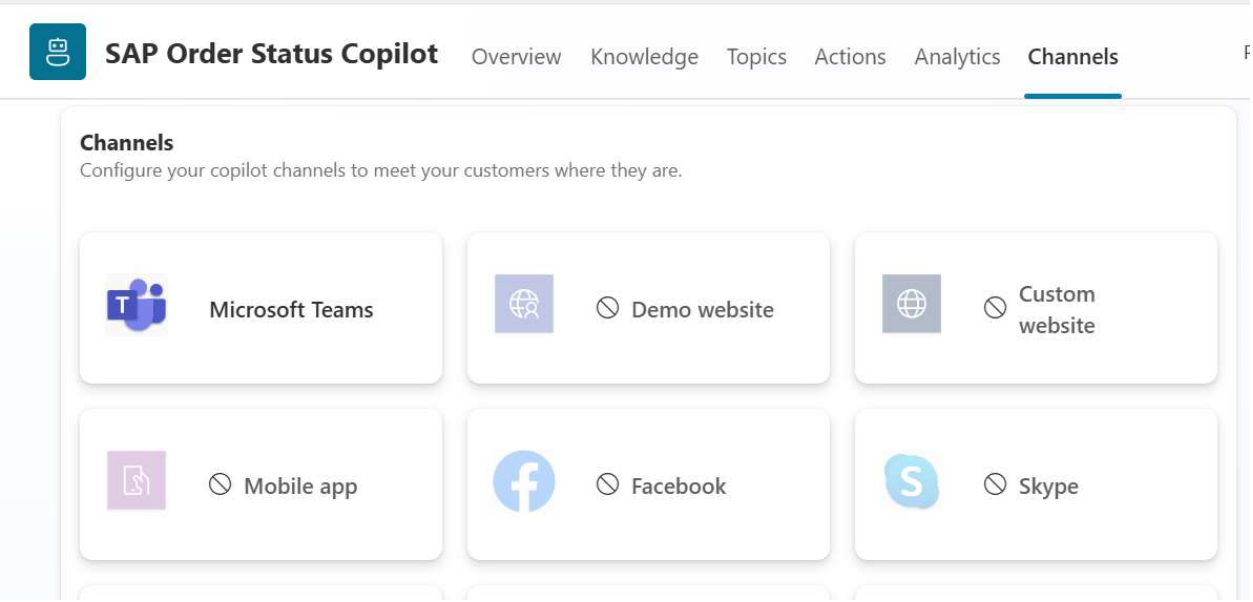


This concludes the flow.

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



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



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.