Steps to recreate the scenario:

Prerequisites:

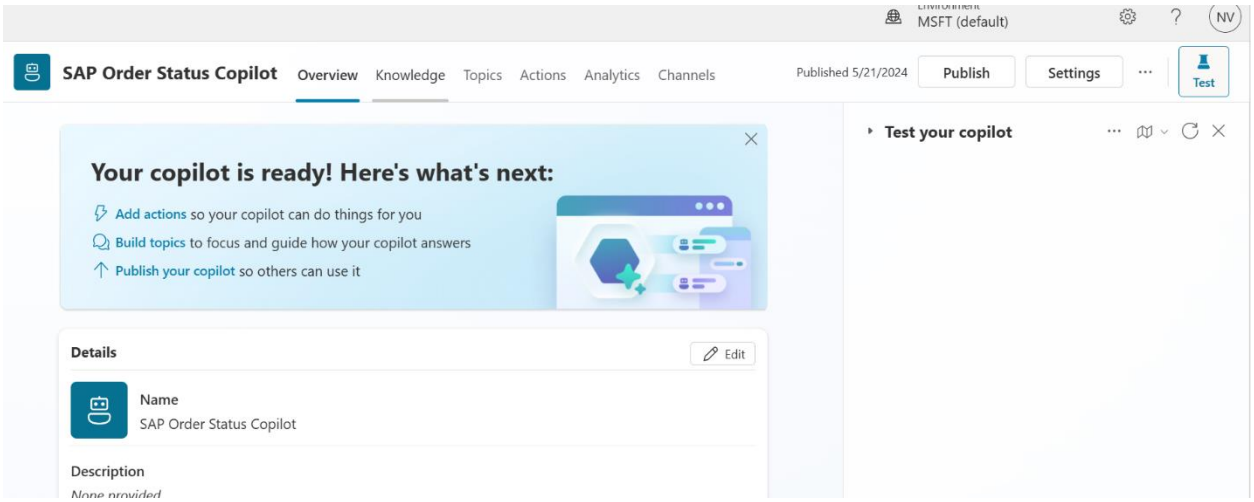
1. **Set up a bot on** Azure AI Studio: For this specific example, the Chat Playground has been used to create and test a bot. You can read more about it here. (Note: Store the URL and the API key for the bot as you will use it later to make HTTP calls to it.)
2. **Set up Microsoft Power Platform**: Create a M365 Developer account. You can read more about it here.
3. **Set up SAP Your SSO – SAP OData connector on Power Automate:**

    - PowerPlatformblog
    - MicrosoftLearnarticle

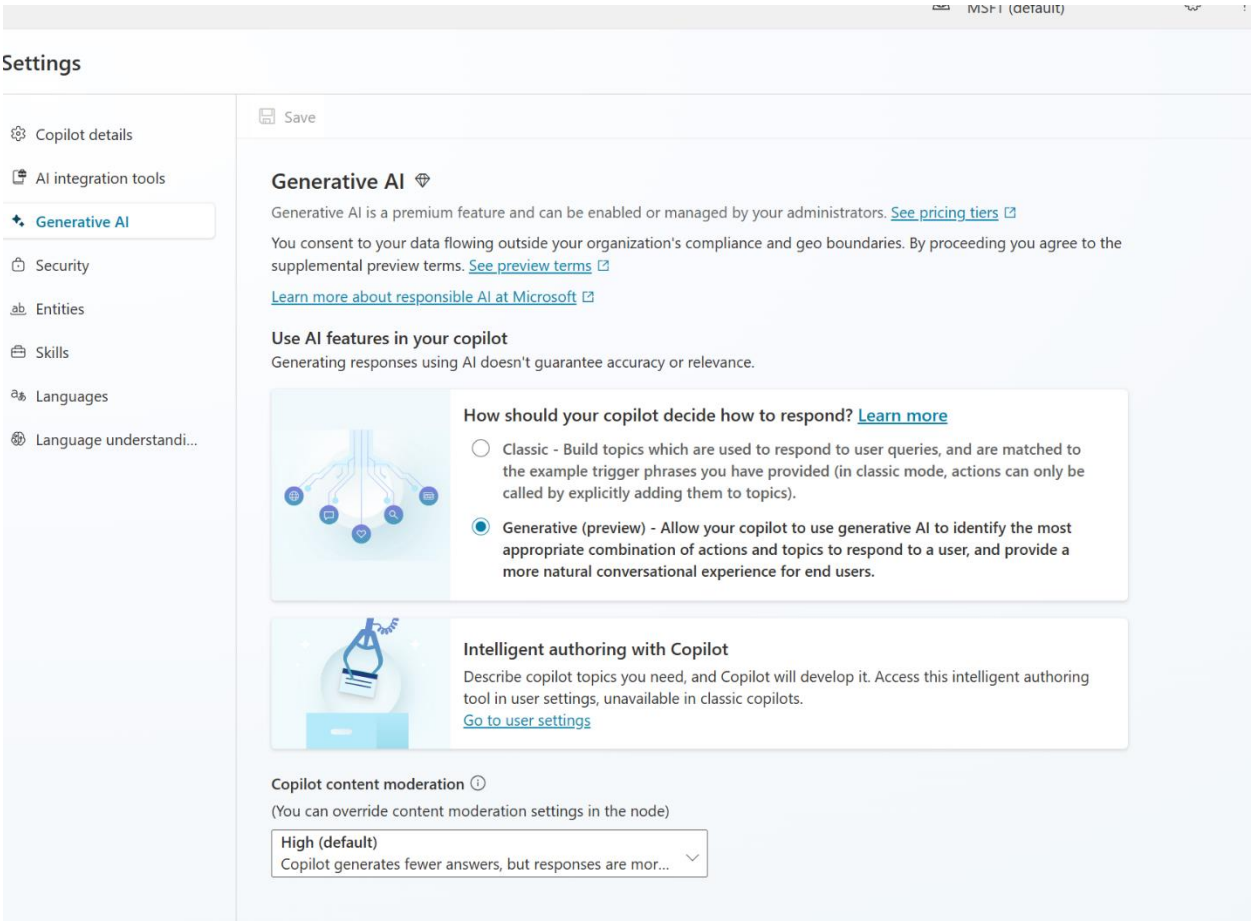
4. **Set up / Sign up for Copilot Studio**. Learn how to get started with it here.

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):



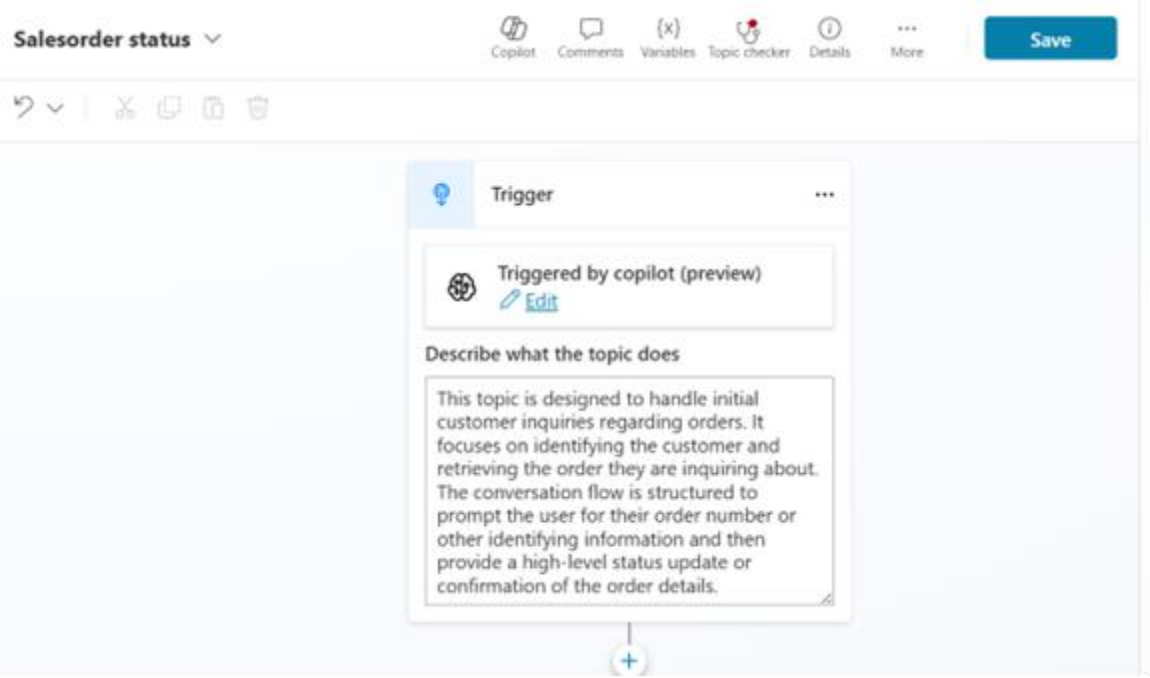
The next step is to create a new topic and the name is – “Sales order status” – this is the topic that the salesperson will use to get the order status and information on the order.

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 a piece of code to call based on the intent of a sentence!

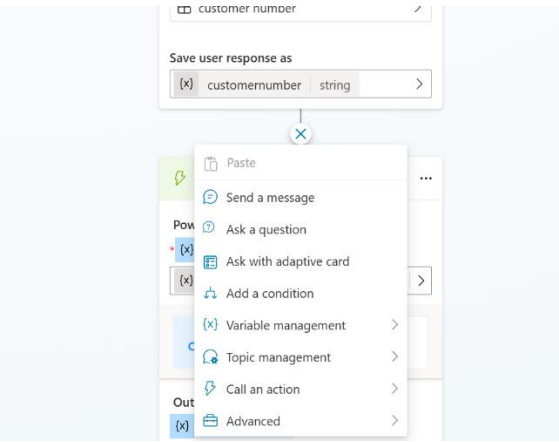
Therefore, the first step after creating a new topic is to write a good description of what topics will do. This is what will help the AI identify when to call this flow.

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]

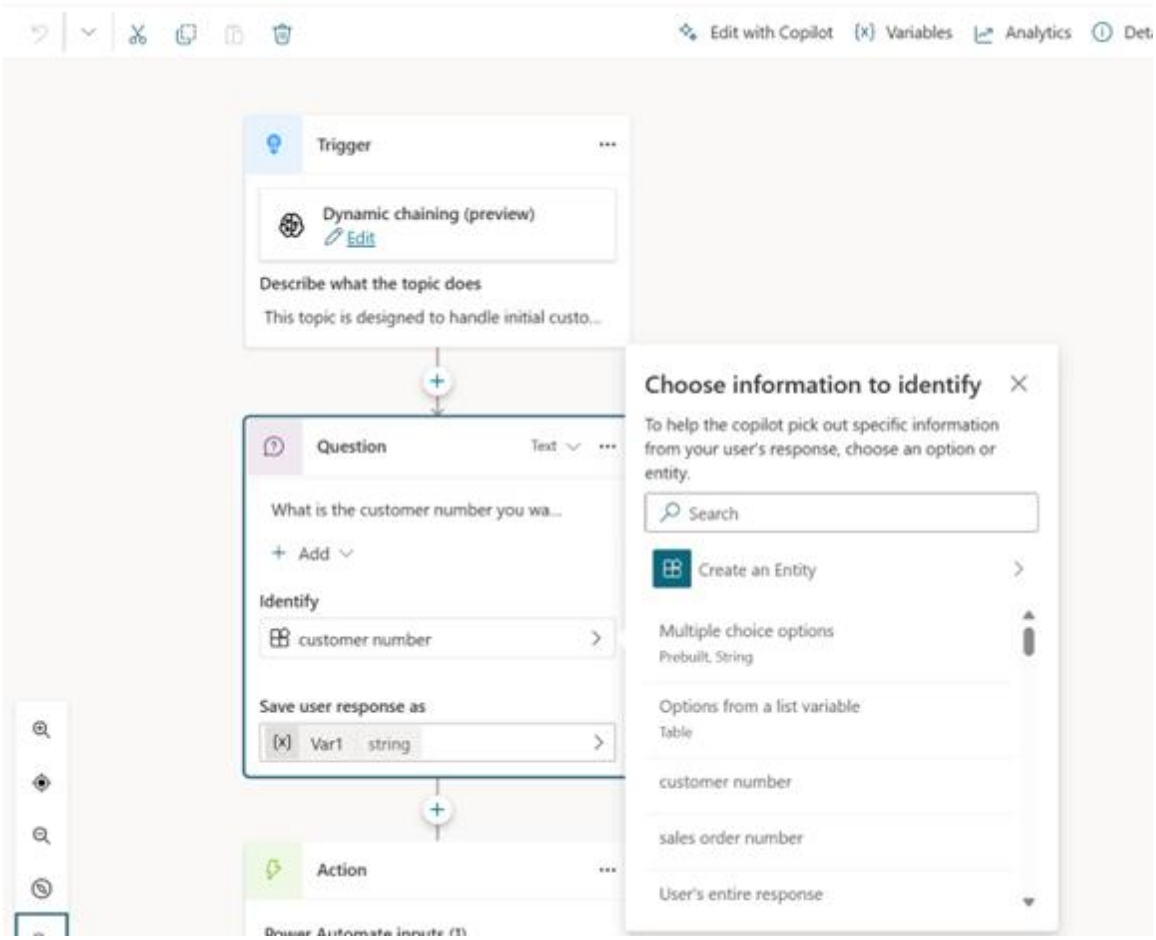
Here is the image of the topic description used for this. (Fun fact- the Microsoft Copilot was used to write the description.)



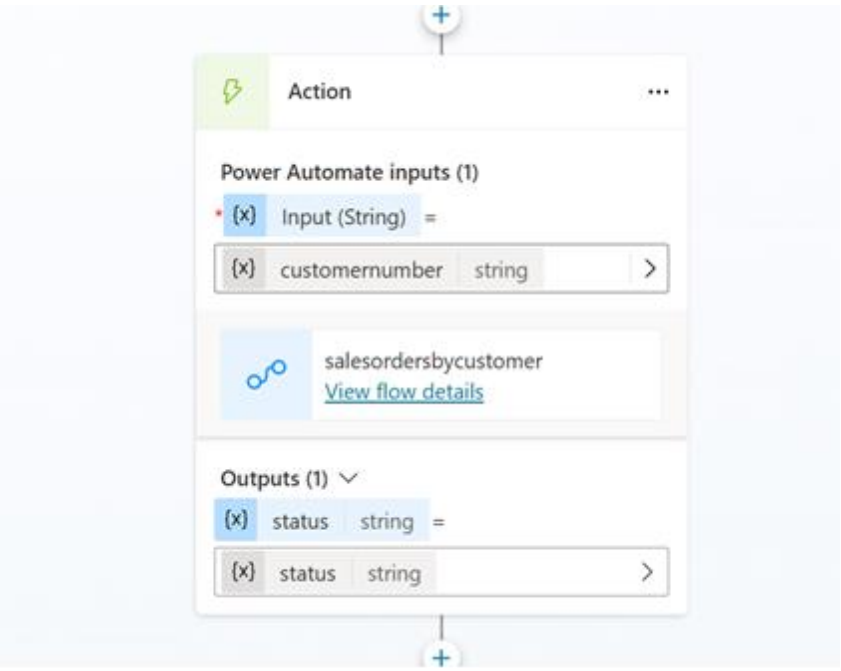
The next step is to add a “question node” to ask the user what customer number they want this information for.



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 for customer number is for your system). Here is a link on learning more about creating entities.

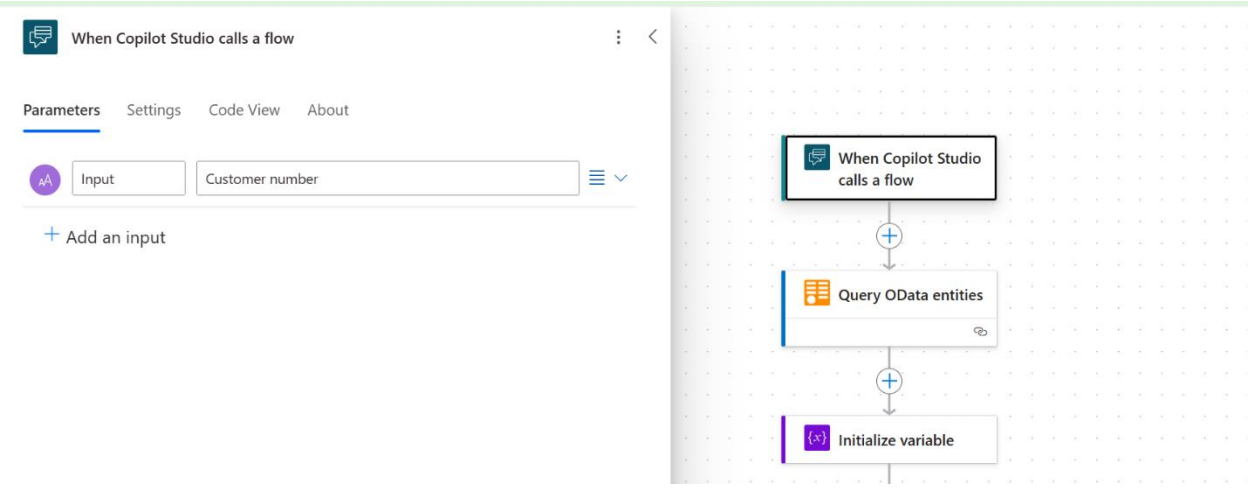


The next step is to now call an action and create a Power Automate flow that gets and parses this information from the SAP system.



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- OData on Github”.

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

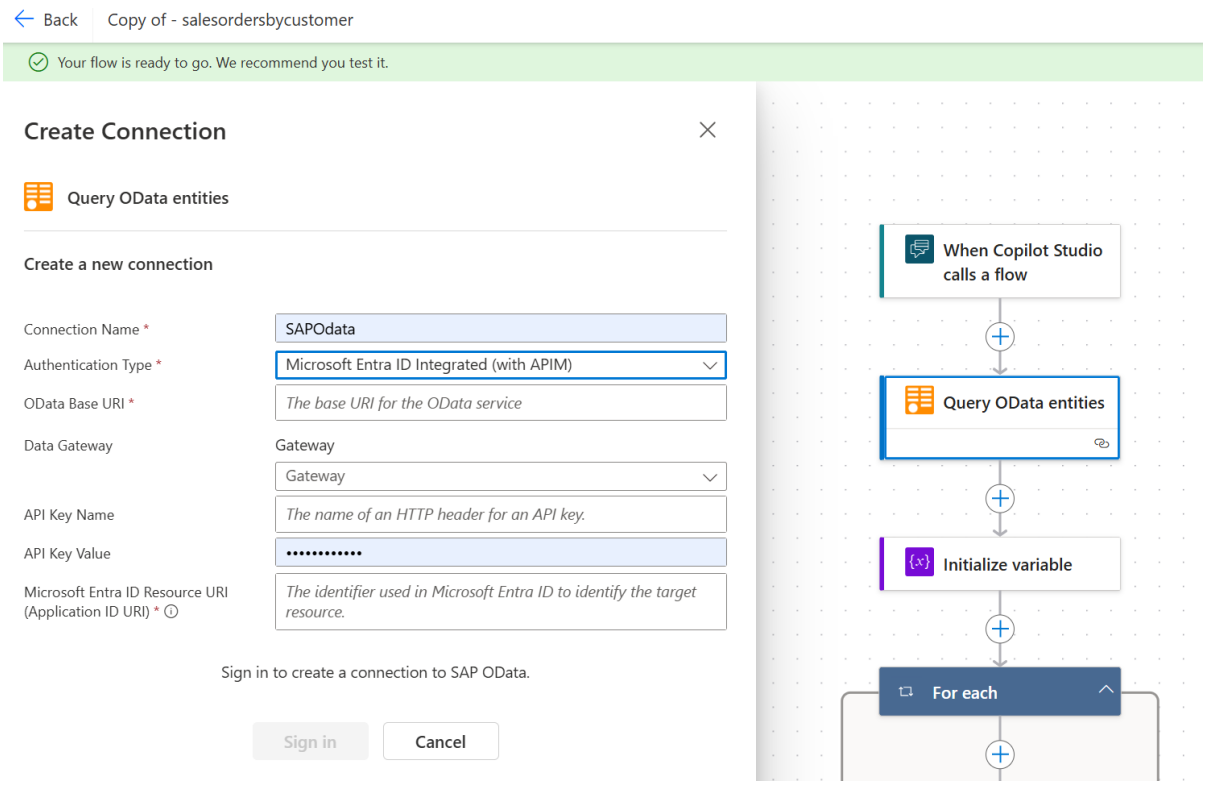


The first step is to set up single sign-on using the OData connector and come up with a base URL for the API_SALESORDER_SRV – Add that base URL into the OData Base URL field and then also add the URL for your application for Entra ID.

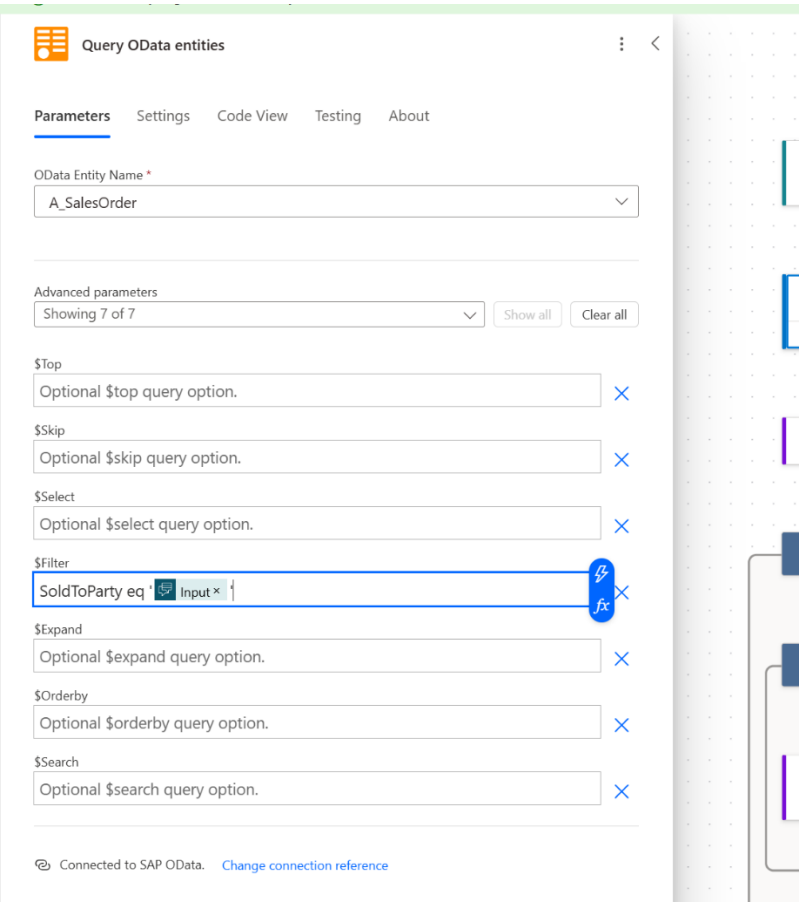
Read the instructions on how to set this up here:

PowerPlatformblog

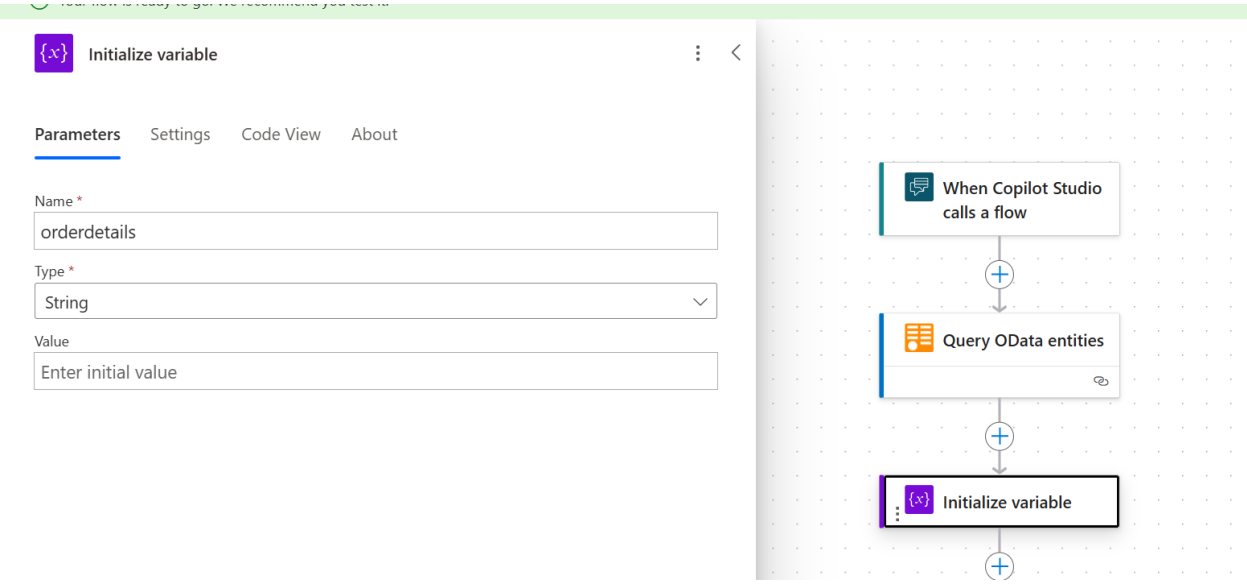
MicrosoftLearnarticle



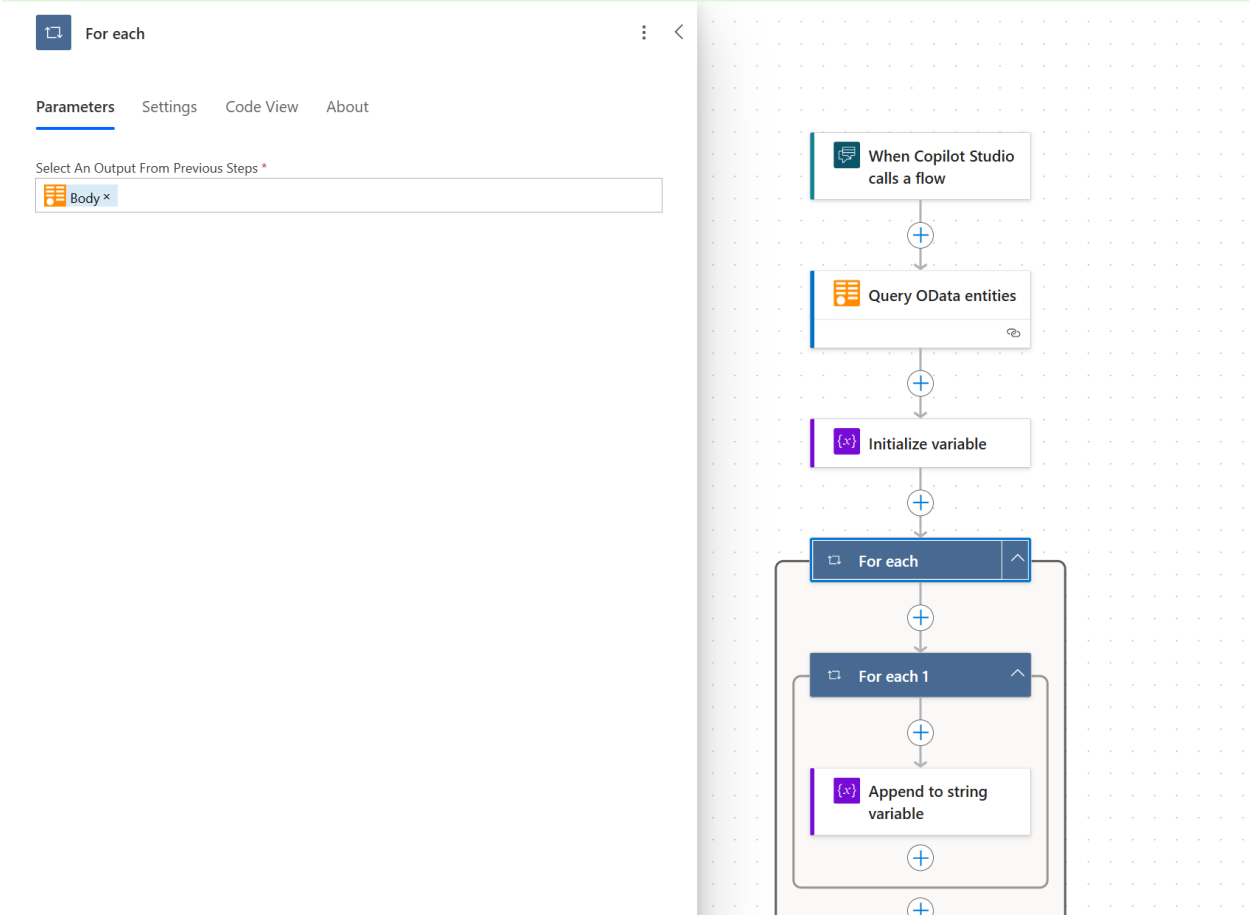
Next from the Entity drop down, select A_SalesOrder, and filter the SoldToParty (which is the customer number) as shown below and make it the input received from the Copilot Studio.



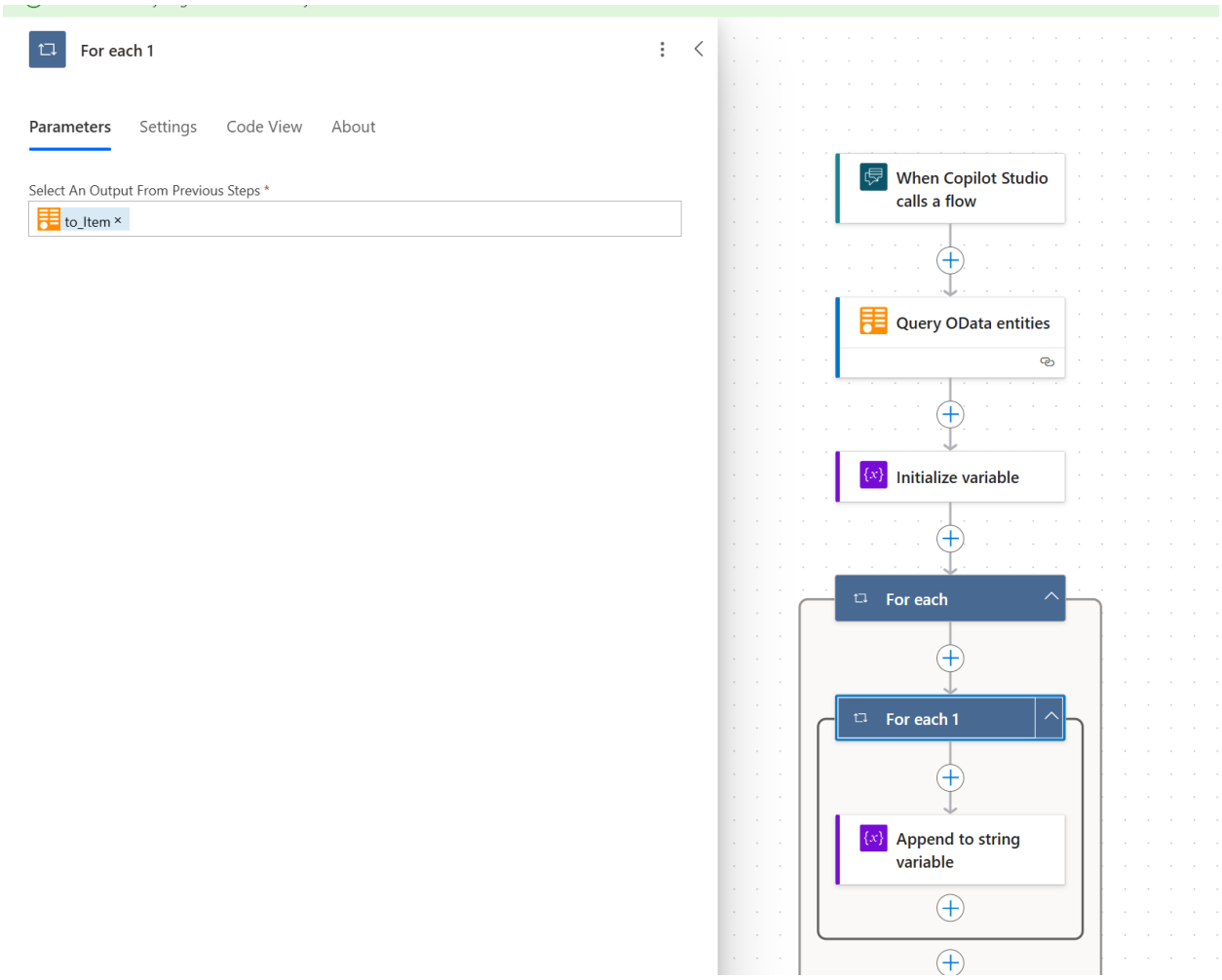
Next, define a variable called ‘orderdetails” This variable will store all the information on the orders for this customer.



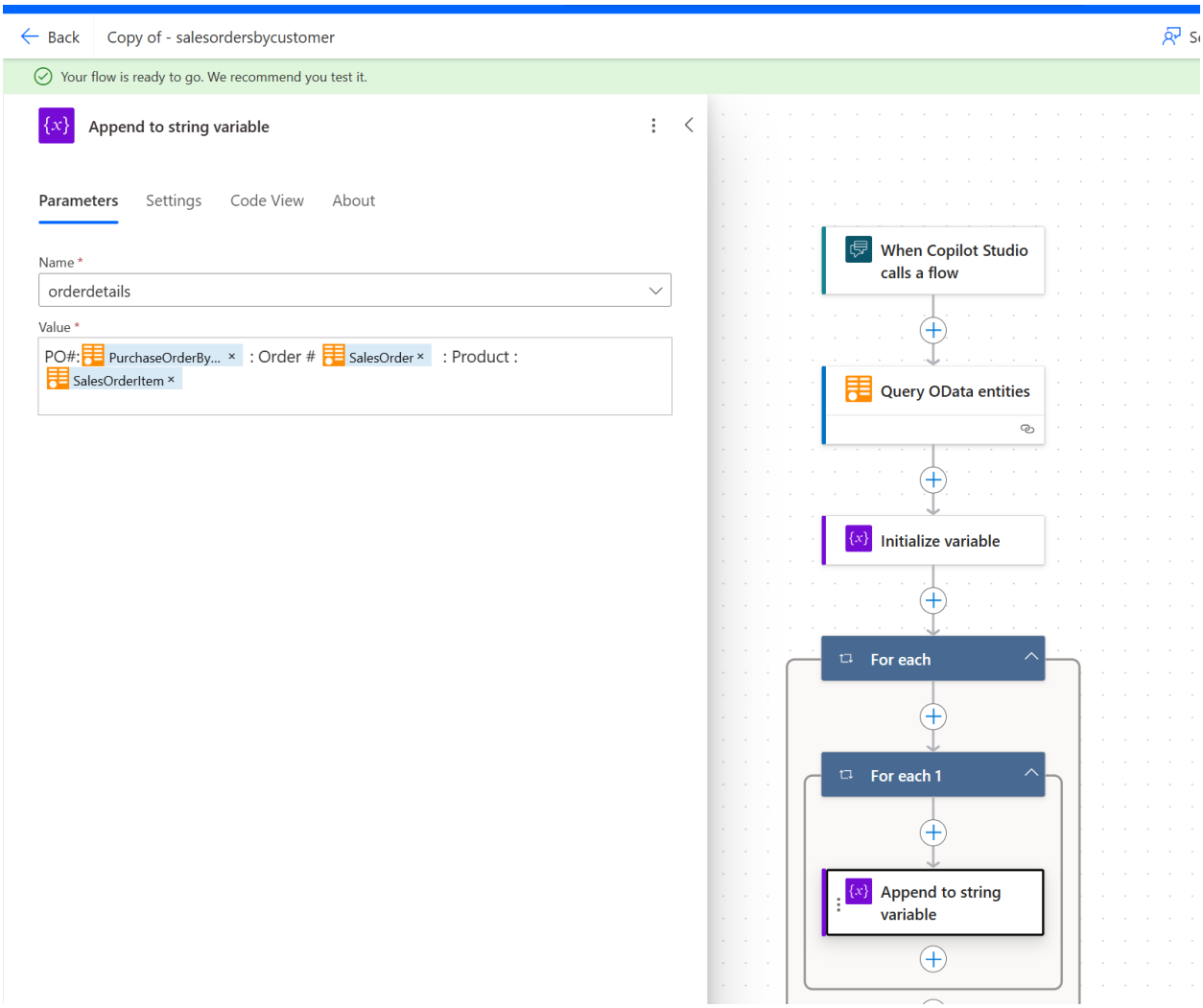
Now we will loop through each sales order for the customer with the OData API call returned as shown below:



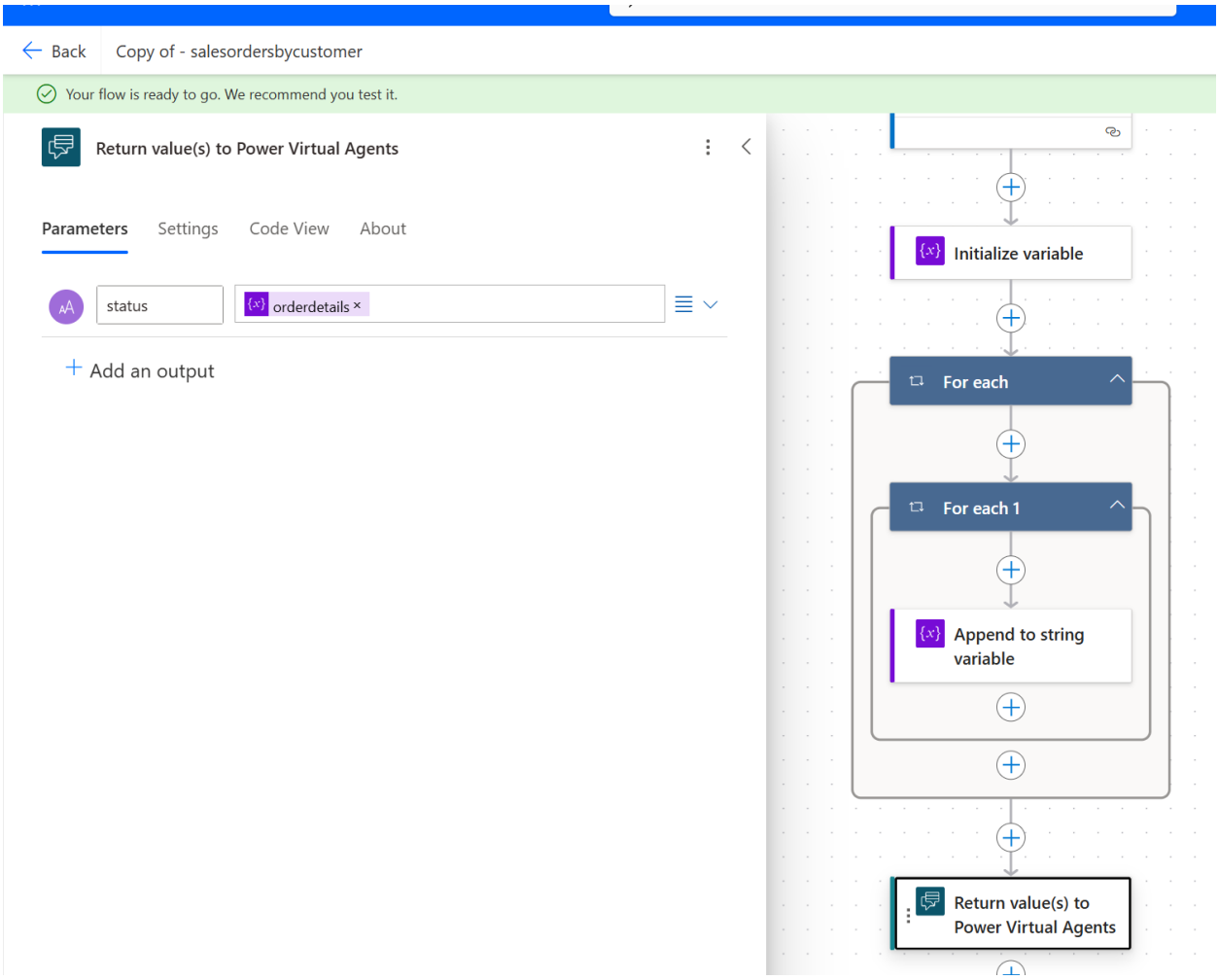
Next, each sales order may have multiple items, so we will loop through each item and add it as a separate entry in our variable.



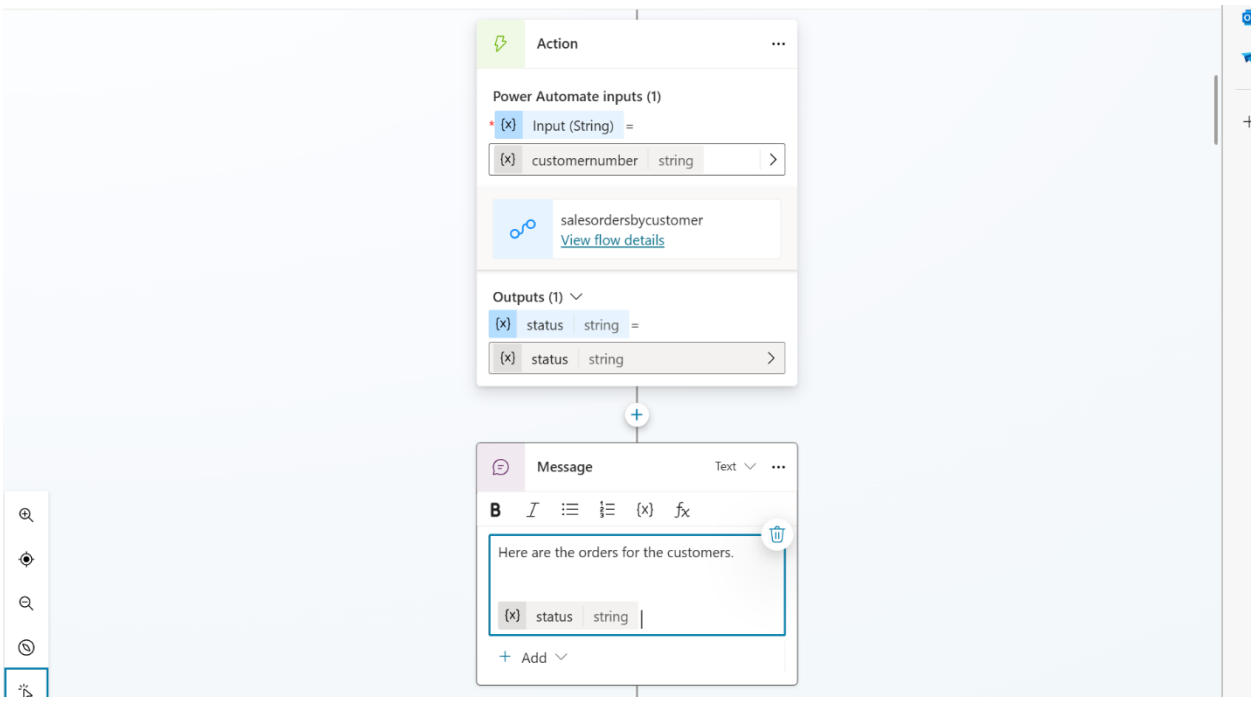
To append to our variable, for each item we will use the dynamic variables in Power Platform and add in these variables.



Finally, after we have information on all the orders for that particular customer, we will return the variable order details to the Copilot Studio.

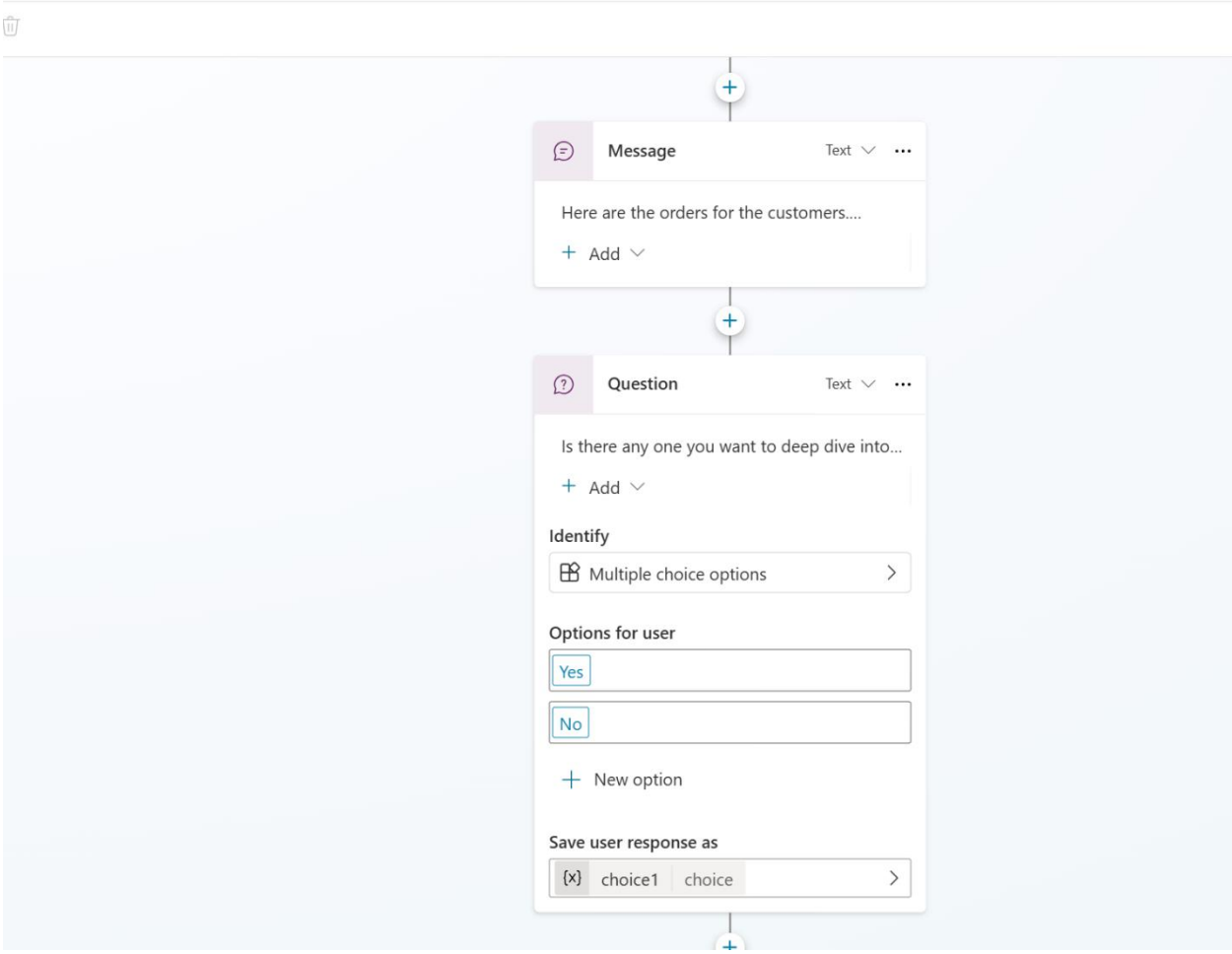


Now back in the Copilot Studio we will append the response returned by Power Automate flow as shown like this.

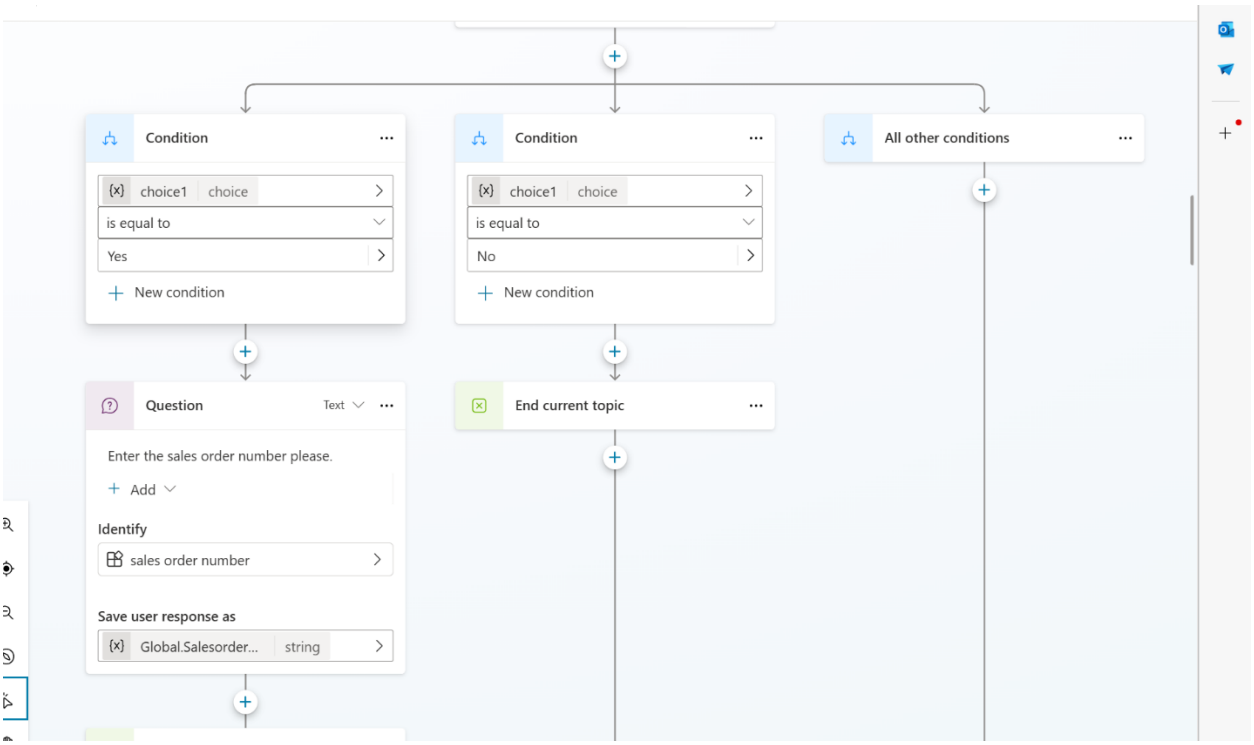


Once we have shown the salesperson all the customers' sales orders, they can now act on it and dive deep into the sales order they want to investigate.

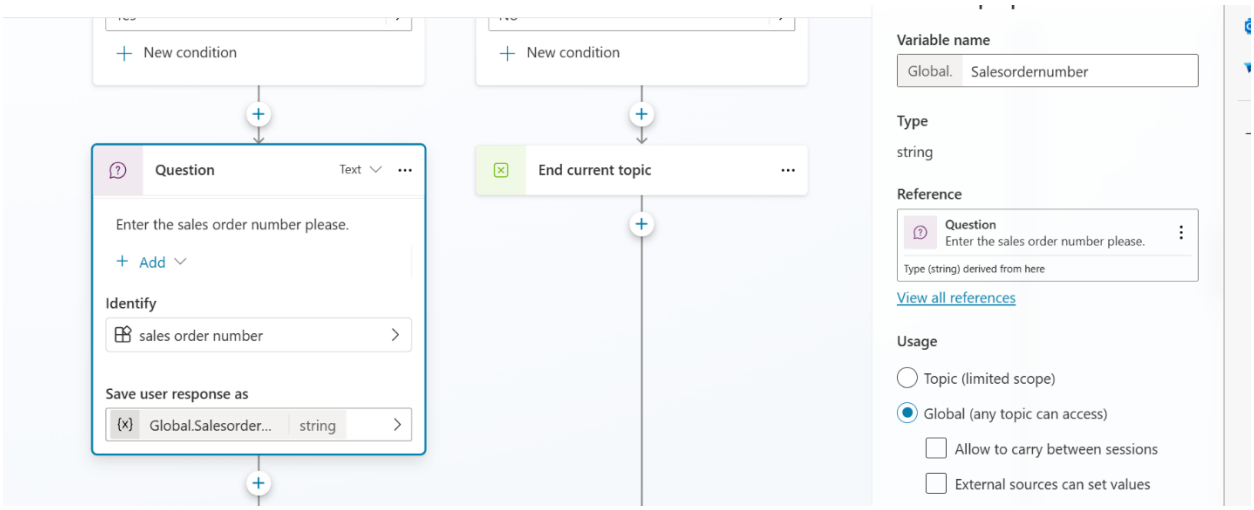
Note: In this demonstration one such action that the salesperson could take has been shown. The action we have selected for this demo is to ask the salesperson if there is any particular problematic order they want to dive into and confirm the status of. However, you could create any such flow based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



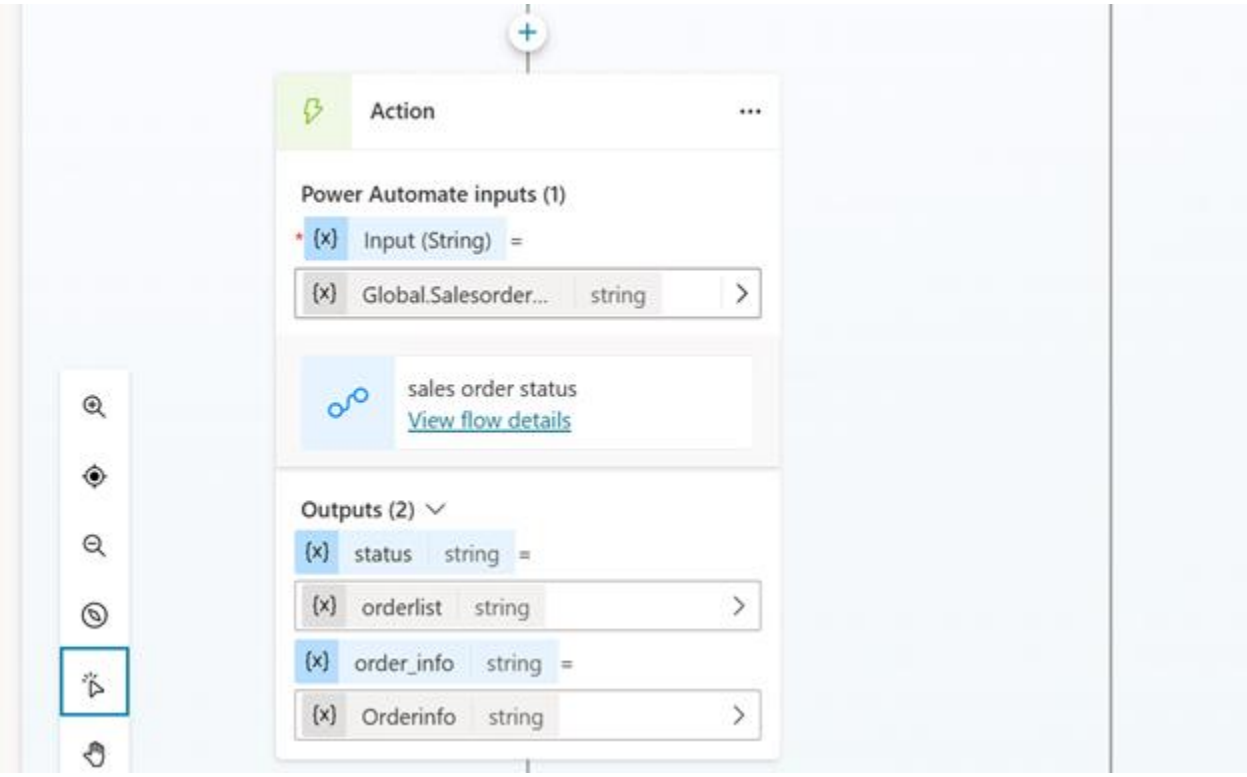
We will now create a question node and ask the salesperson to enter the order number in the Copilot Studio. We will once again create a new entity with the regex in the format of the SAP sales order number (the same way we did it for the customer number above).



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 to make it a global variable.



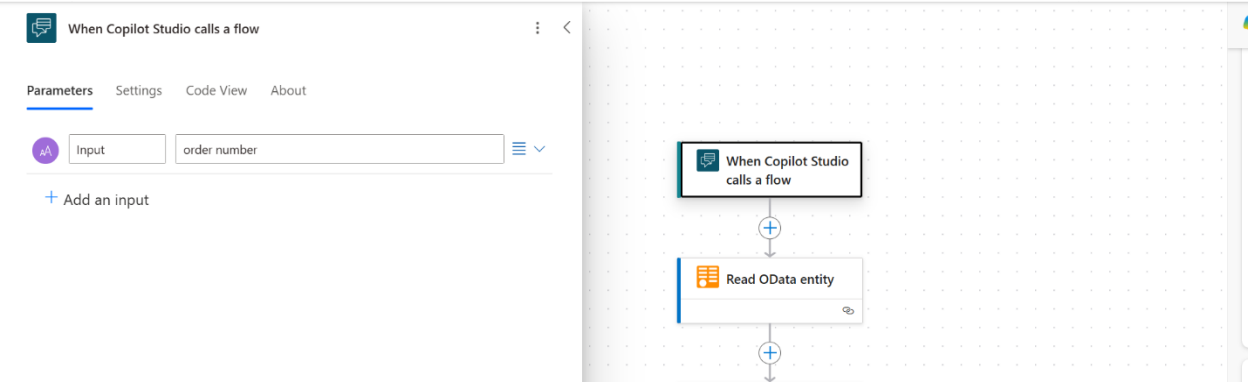
Once we have the sales order number that the customer wants to use, we will now dive deep into this sales order, find out about its status from the SAP system and find out whether it is delayed or not.



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.

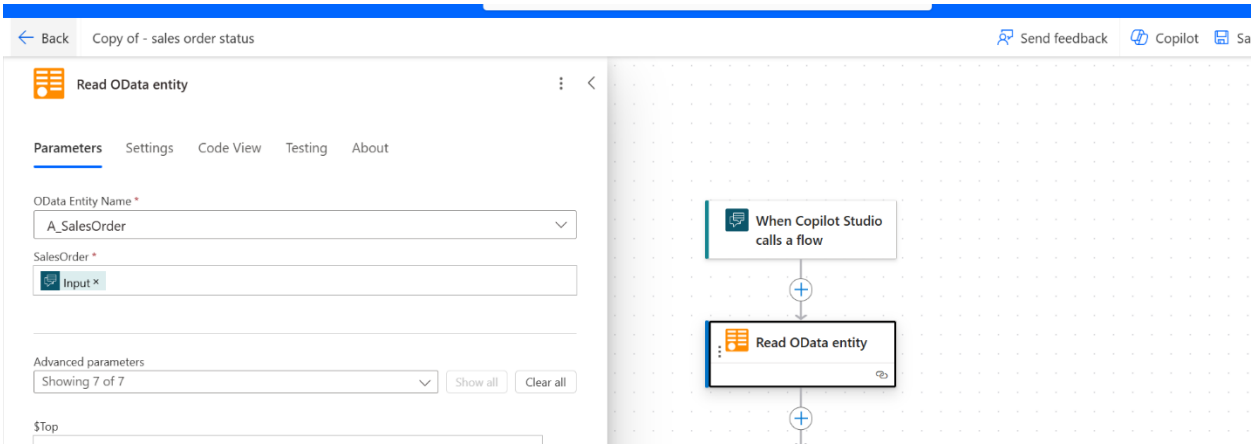
Here are the steps for this Power Automate flow “sales order status- Odata.”

First the input to the Power Automate flow will be the order number to find the status for.

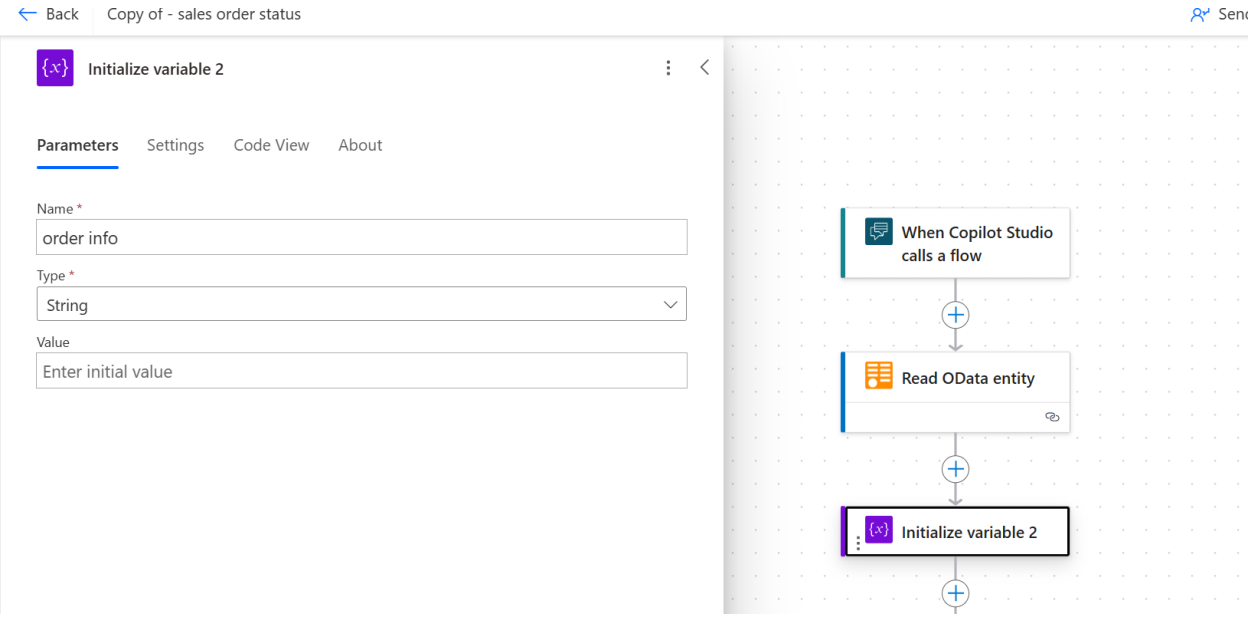


We this time, select the “Read Entity” SAP OData connector and once again set up the OData connection using Entra ID as the log in and the same APIM URL for API_SALESORDER_SRV will be used as the previous Power Automate flow.

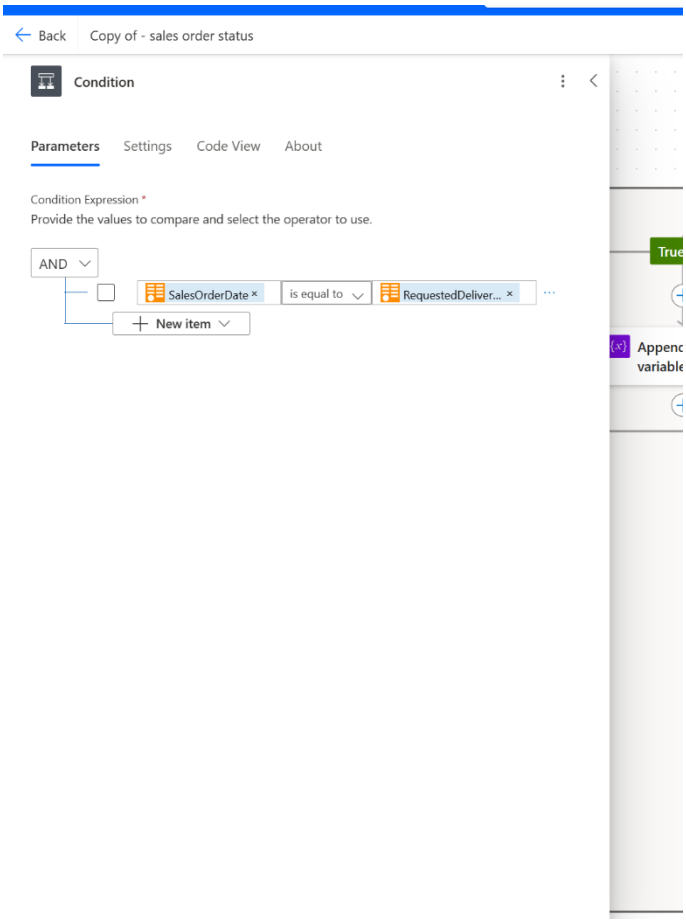
In the “SalesOrder” field, the input sales order number will be put to retrieve the required sales document.



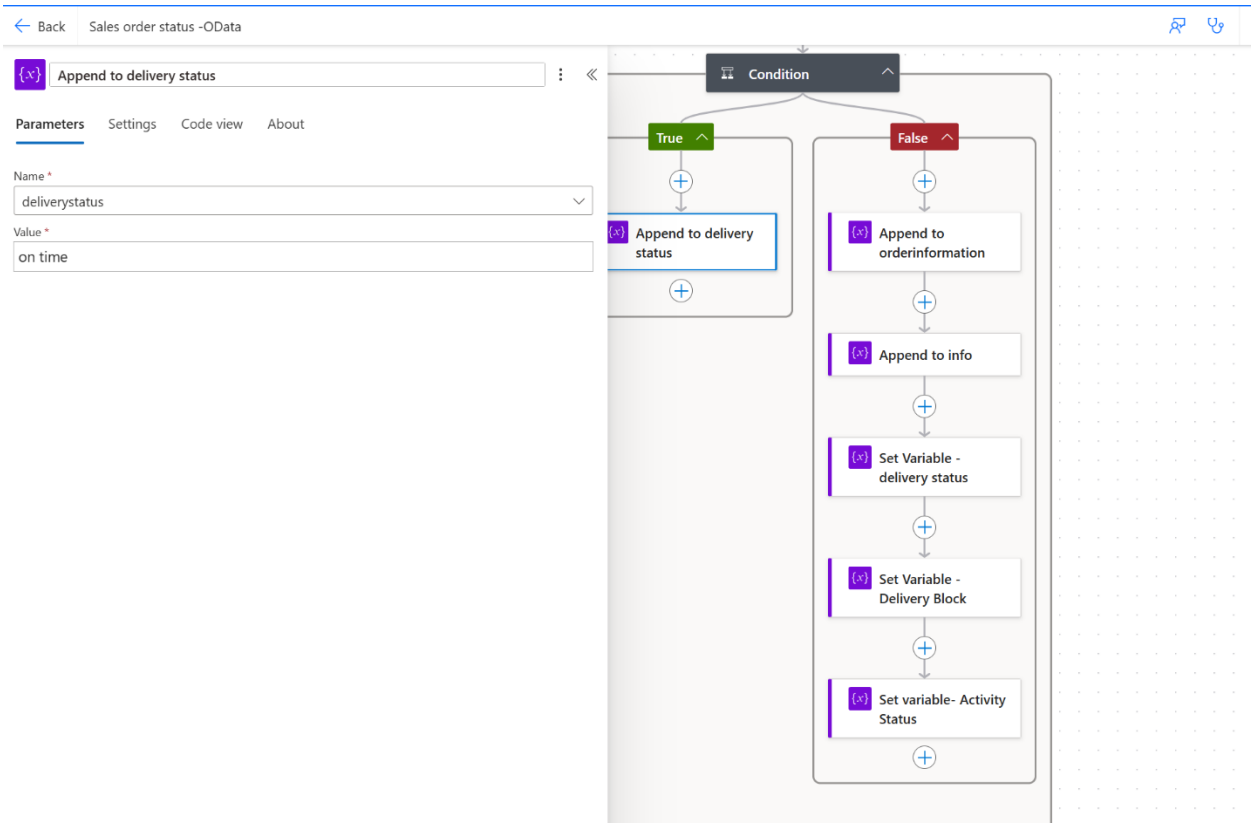
Next a variable to store the order information called ‘order info’ will be created along with two other string variables called “Delivery status”, “Delivery Block” and another one called “order information”. These variables will store all the information about the sales order that would be returned to the Copilot Studio.



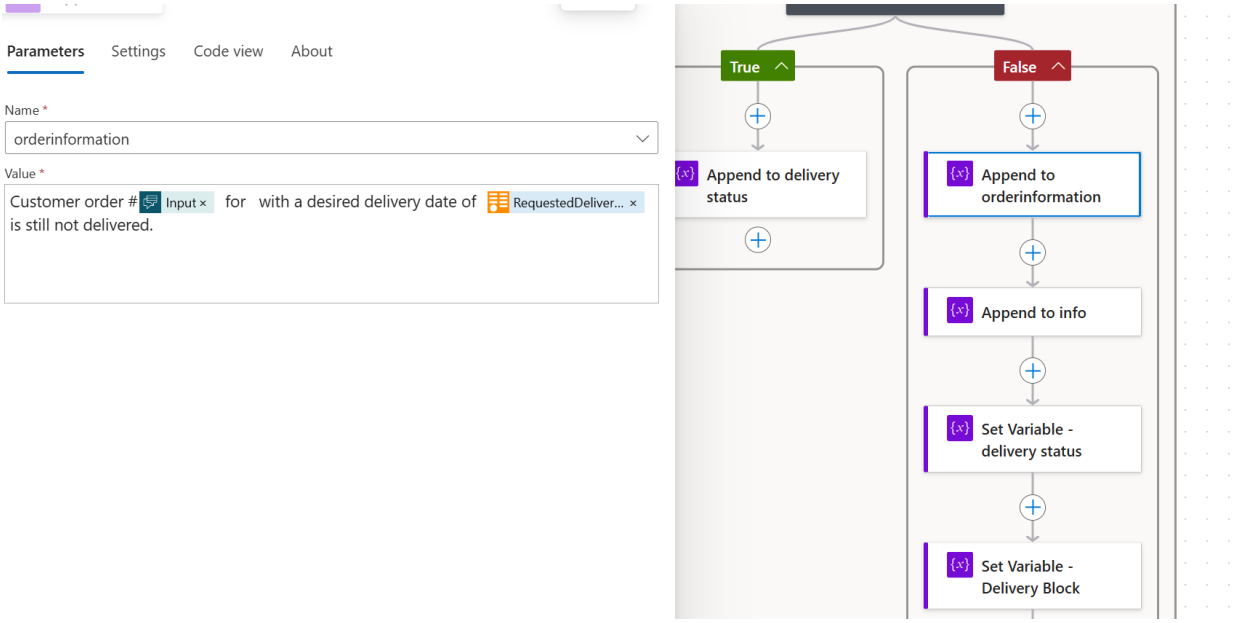
After those variables are initialized, we will now check whether the order was delayed or not from the OData response, this can be checked by using the Dynamic variables “SalesOrderDate” and “RequestedDeliveryDate”. If the requested delivery date and SalesOrderDate are the same, the order was on time, else the order would be marked as delayed.

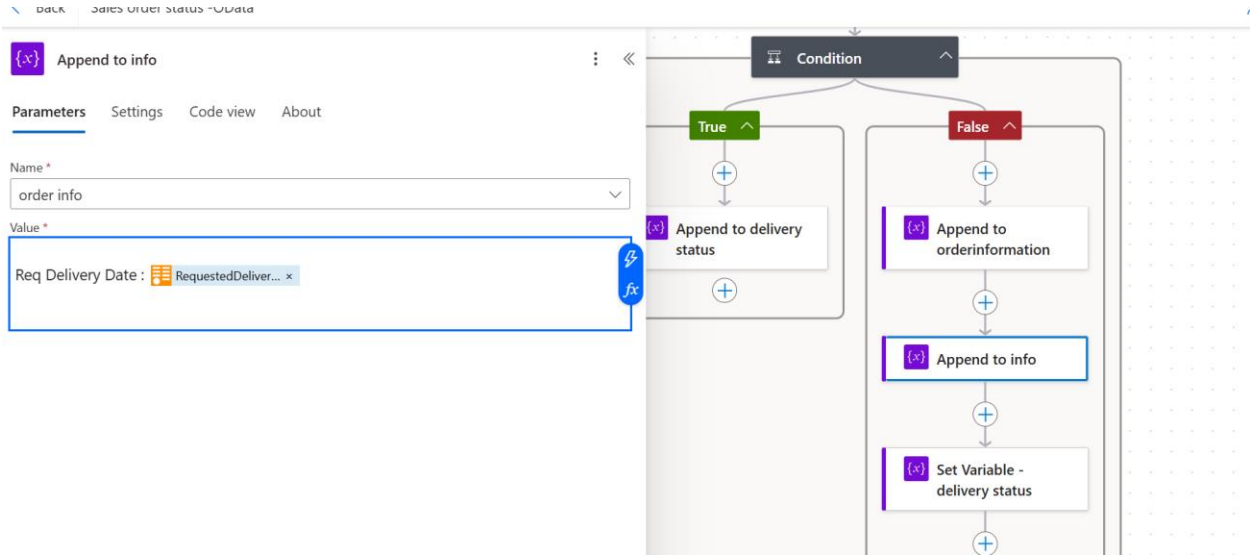


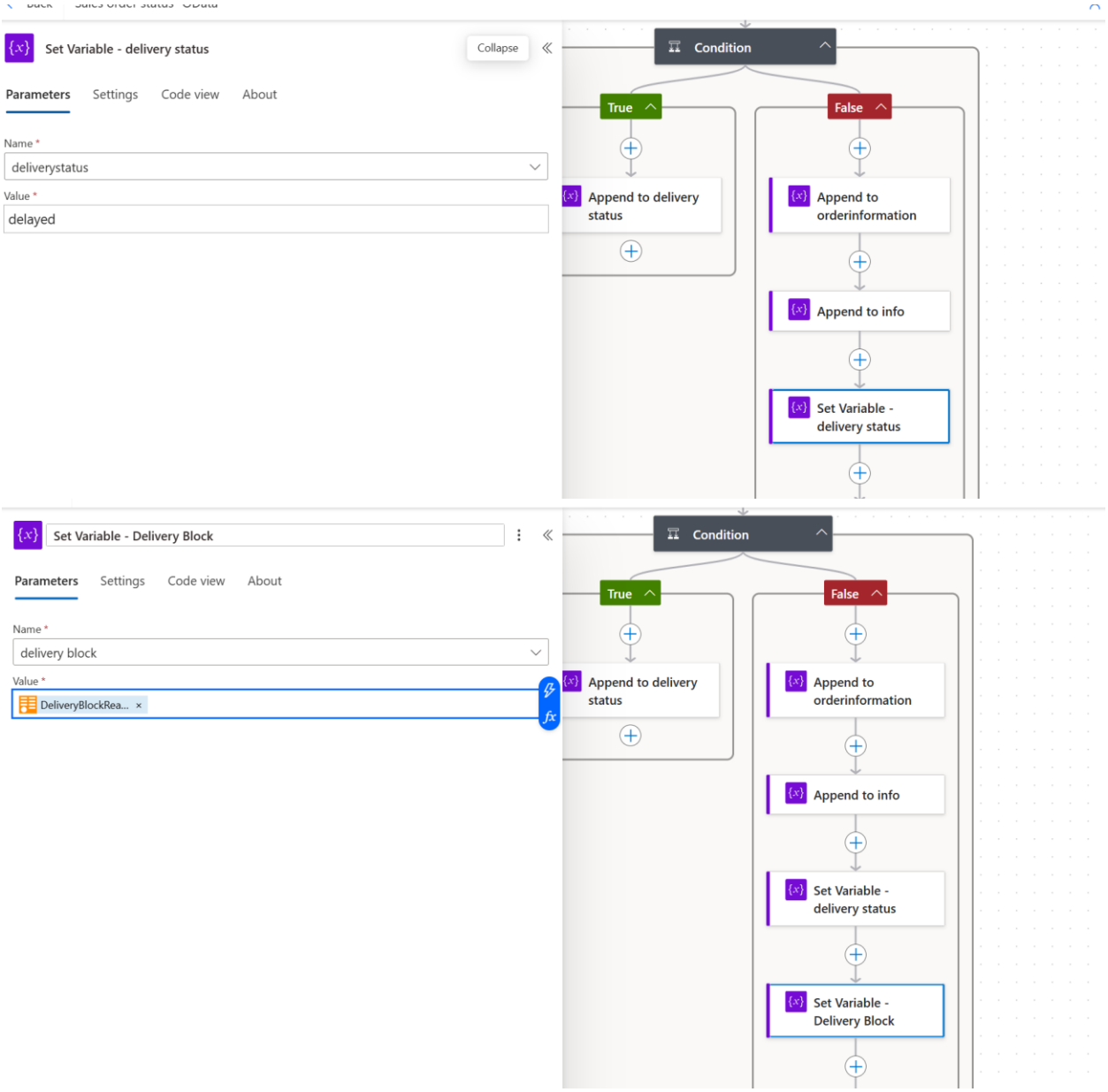
In the condition that those values are the same and the order is on time, we will change the ‘Delivery status’ of the order to “on time”



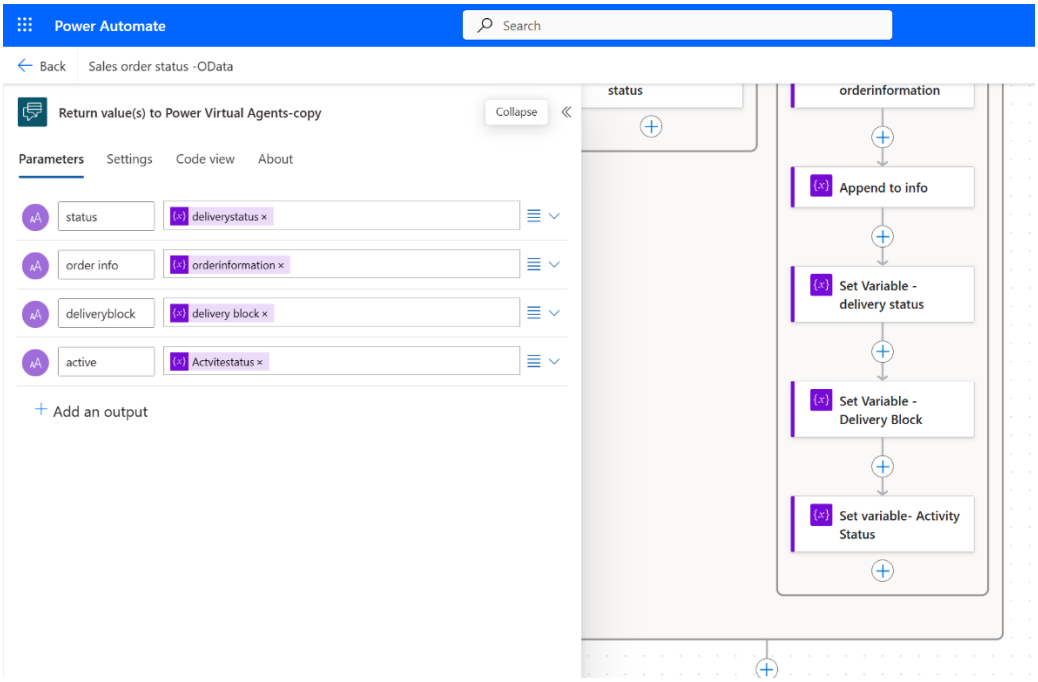
In the condition that they are not the same, we will append the strings as shown below to the different variables.


Parameters
Settings
Code view
About
Name *
orderinformation
Value *
Customer order # Input × for with a desired delivery date of RequestedDeliver... ×
is still not delivered.
True
False
Append to delivery status
Append to orderinformation
Append to info
Set Variable - delivery status
Set Variable - Delivery Block

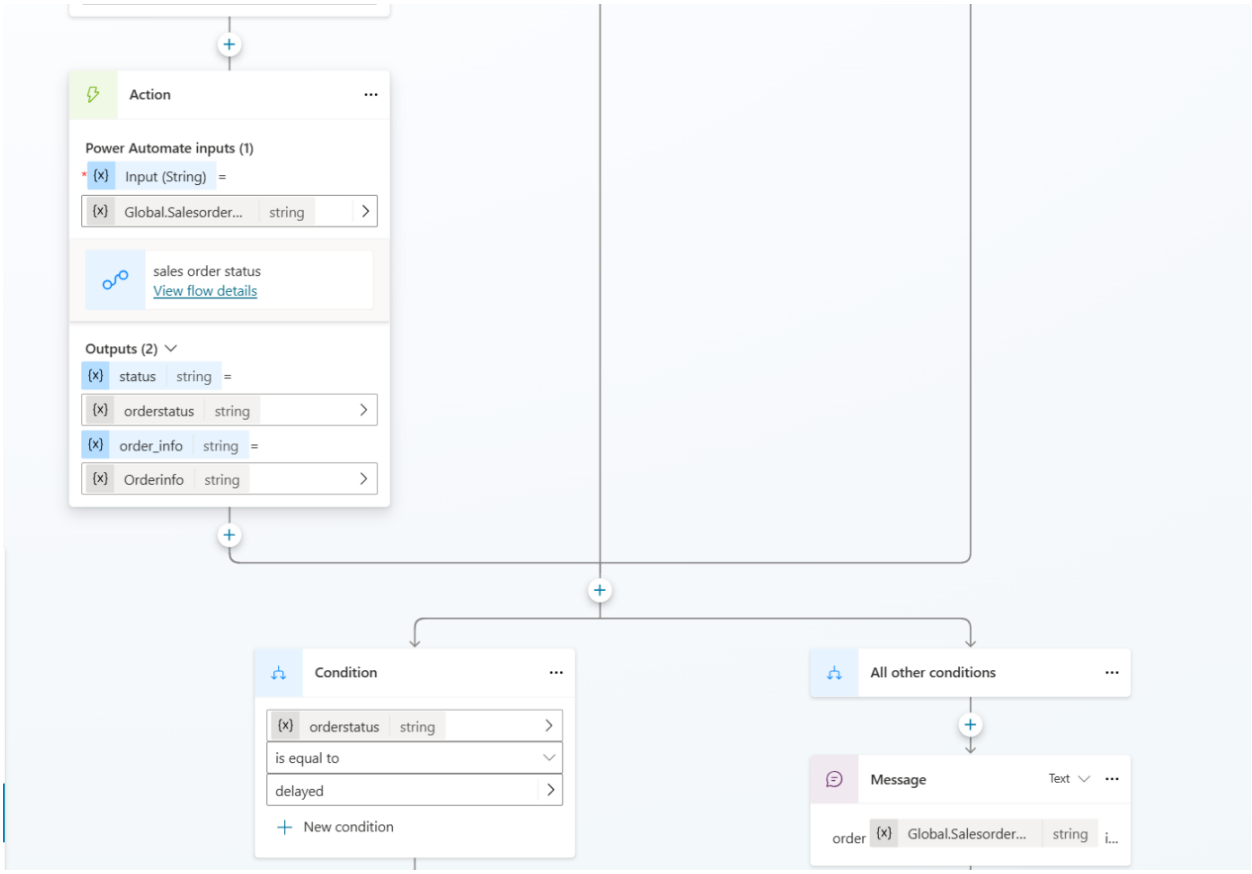

Append to info
Parameters
Settings
Code view
About
Name *
order info
Value *
Req Delivery Date : RequestedDeliver... ×
Condition
True
False
Append to delivery status
Append to orderinformation
Append to info
Set Variable - delivery status

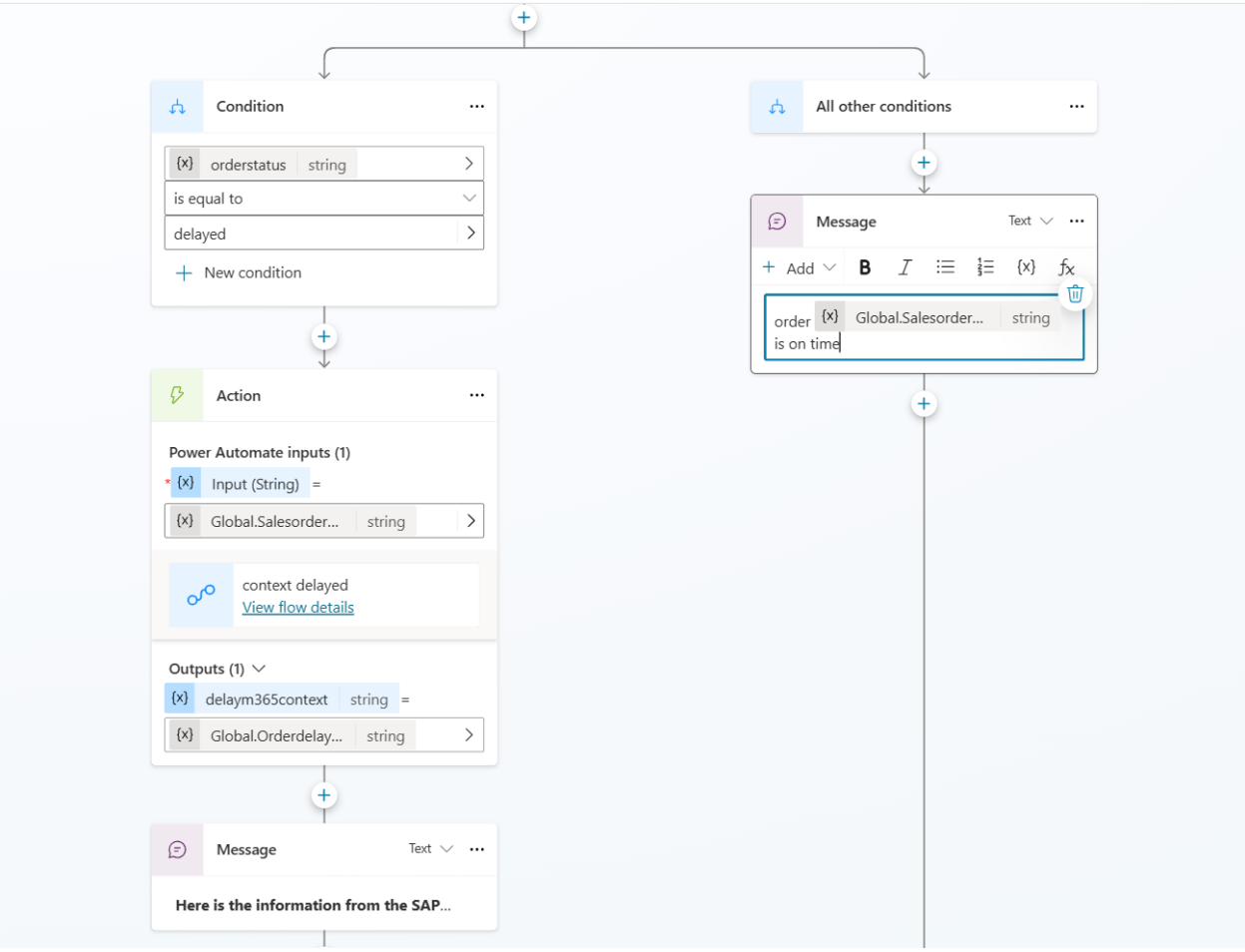

Set Variable - delivery status
Collapse
Parameters
Settings
Code view
About
Name *
deliverystatus
Value *
delayed
Condition
True
False
Append to delivery status
Append to orderinformation
Append to info
Set Variable - delivery status
Set Variable - Delivery Block
Parameters
Settings
Code view
About
Name *
delivery block
Value *
DeliveryBlockRea...
Condition
True
False
Append to delivery status
Append to orderinformation
Append to info
Set Variable - delivery status
Set Variable - Delivery Block

Finally, we will return all these variables to 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.


Action
Power Automate inputs (1)
Input (String) =
Global.Salesorder... string
sales order status
View flow details
Outputs (2)
status string =
orderstatus string
order_info string =
Orderinfo string
Condition
orderstatus string
is equal to
delayed
New condition
All other conditions
Message
Text
order
Global.Salesorder... string

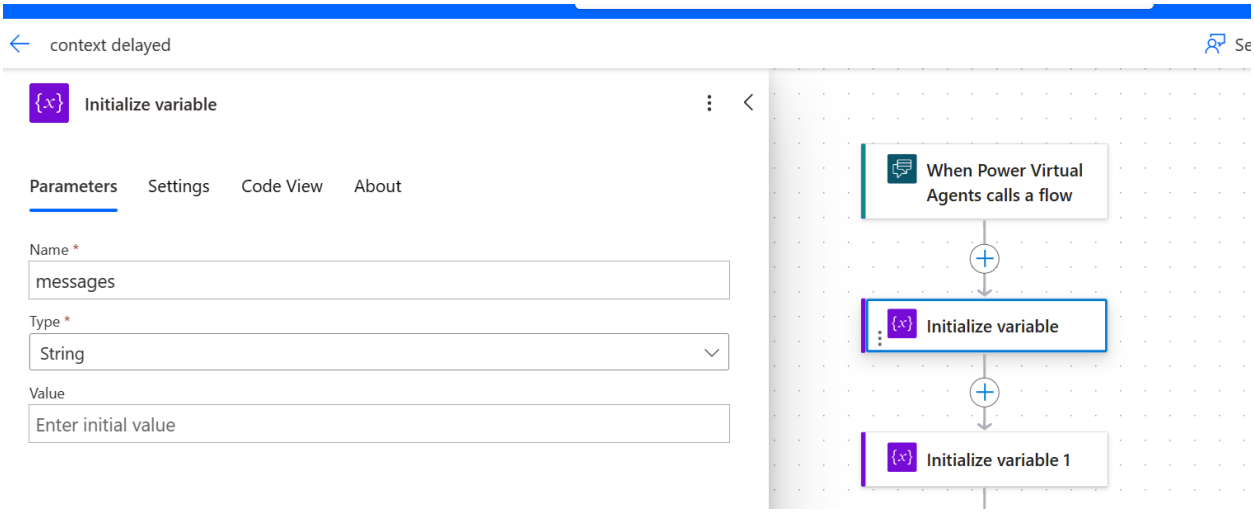


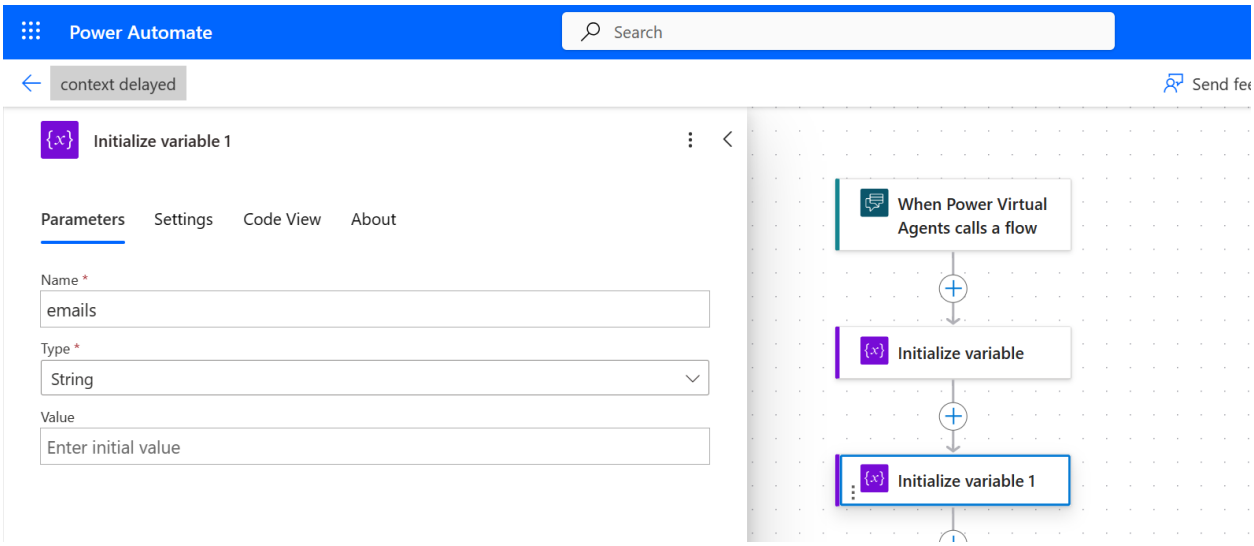
If the order is delayed, the next step is to harness the power of AI and the MS365 data for conversation context (emails and Teams' chats on why the order is late). This is done by querying the MS365 graph for Outlook emails and Teams' chats related to the order and then passing those emails and messages 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”:

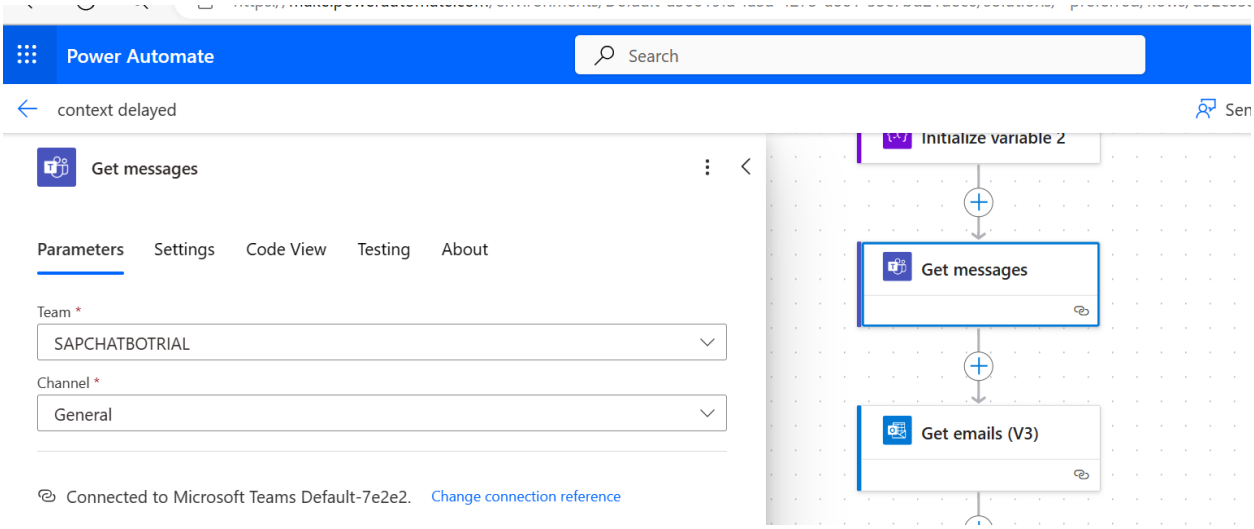
The input for this flow will be the sales order number.

We will first define two variables: “messages” and “emails”.

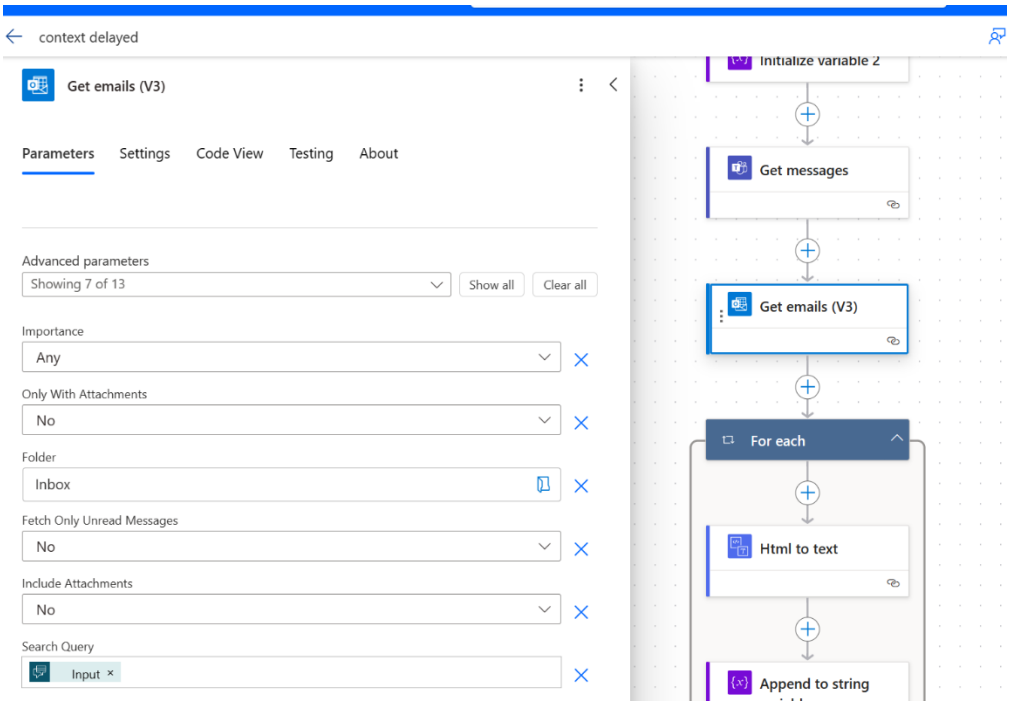




Next, we will also make use of the Team’s connector as shown below, however, the Teams connector does not yet allow you to query messages based on parameters, therefore there will be a bit more processing on that front as shown below, for now we will just fetch all the messages.



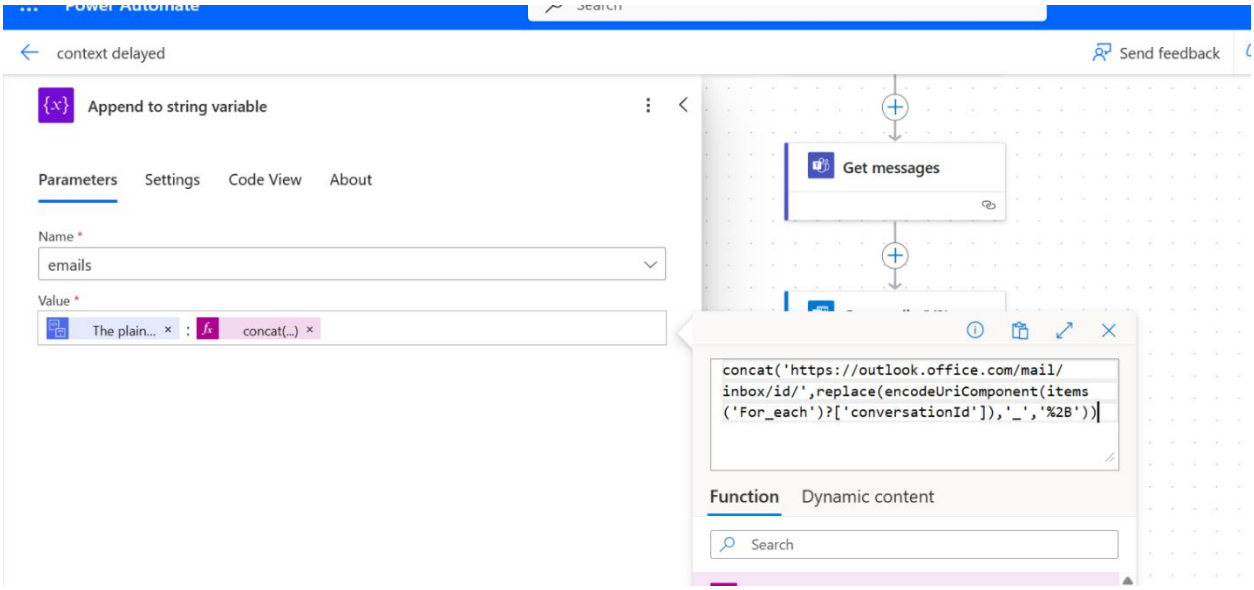
Next, we will fetch all the outlook emails related to the order using the order number input as the search query.



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 for each 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.

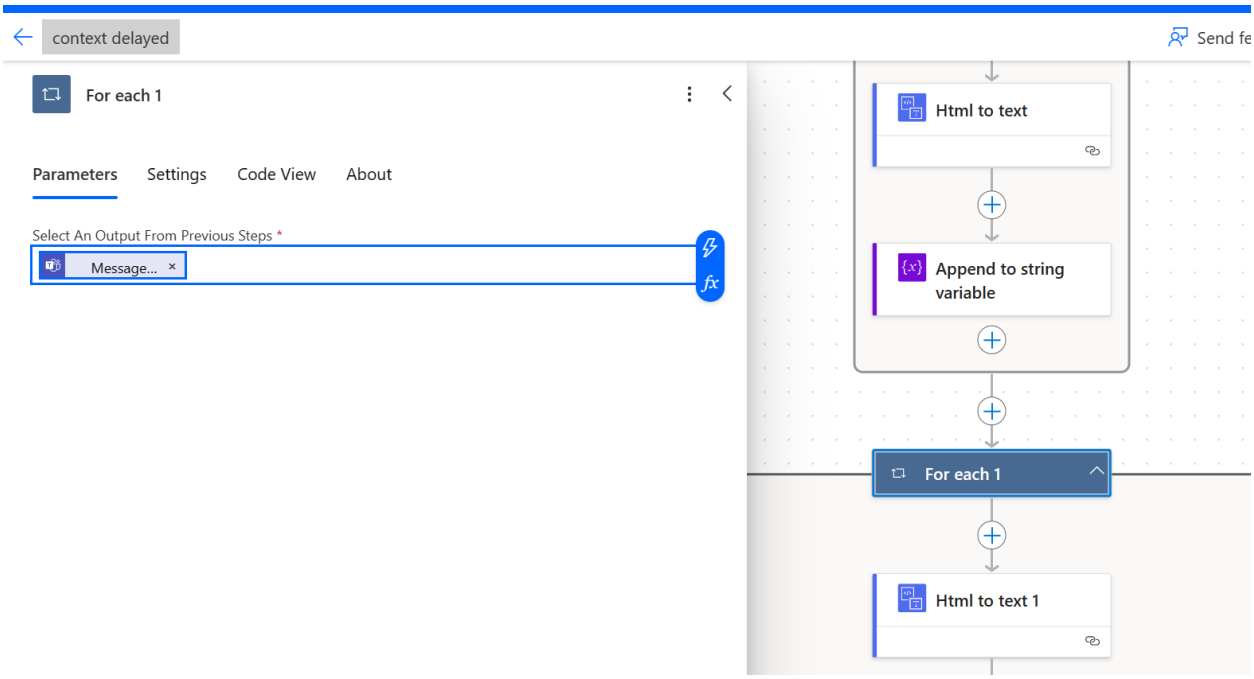
After we get the plain text, we will append it to the email's variable along with the URL of the email, follow the format of the URL as below or refer to the exported flow on GitHub.



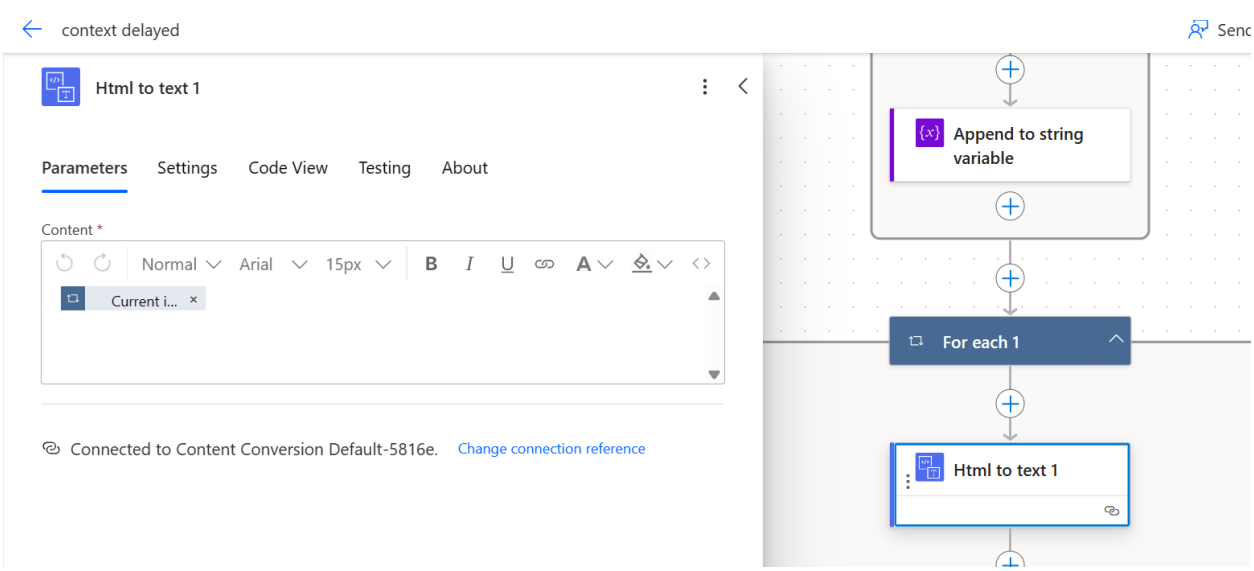
Once we have gotten the emails, we will now work on getting the Teams’ messages.

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.

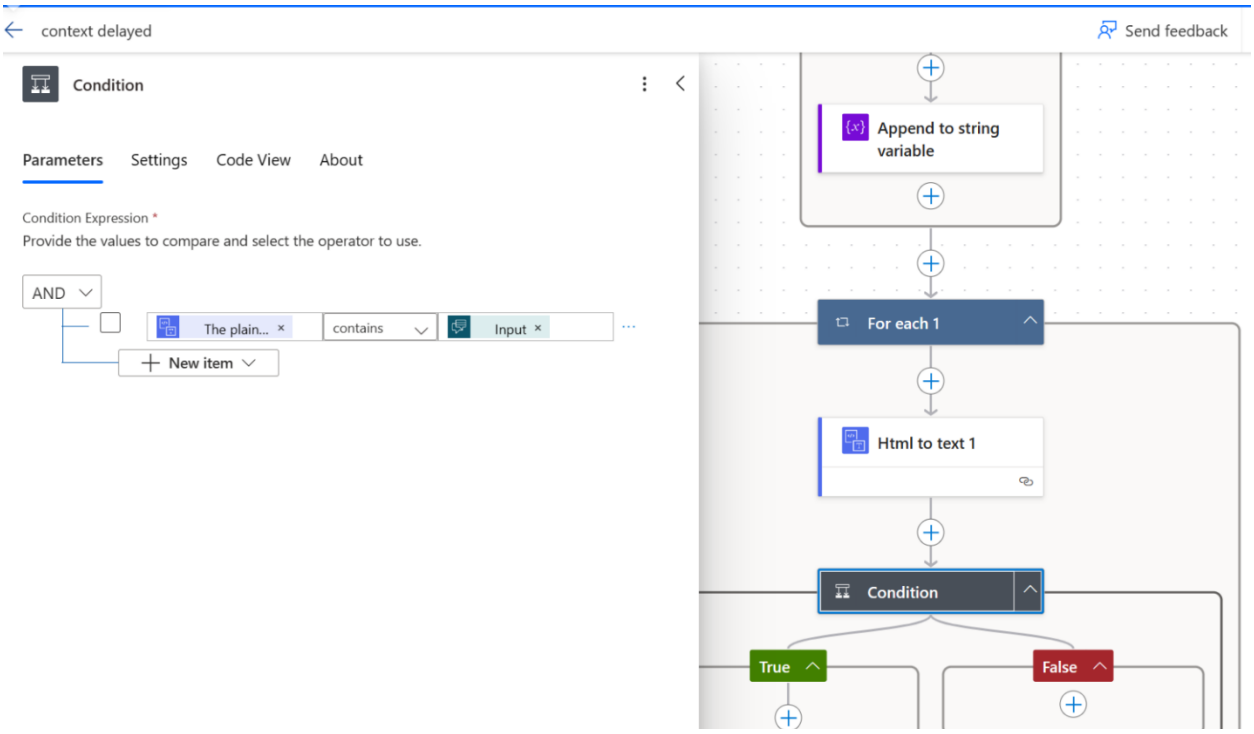
To do this we will loop over all Teams’ messages:



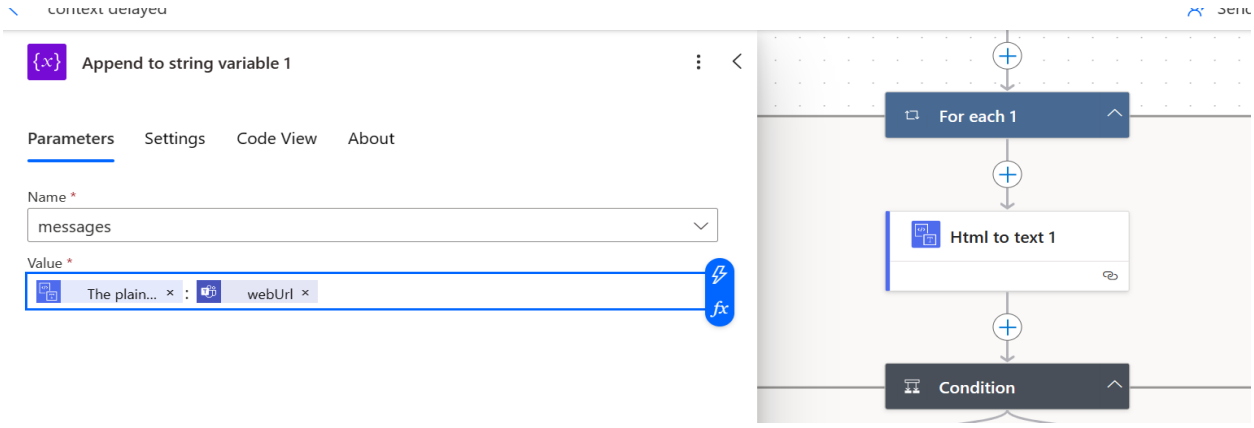
Next, we will convert the response from the HTML format to text using the “Html to text” action.



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).

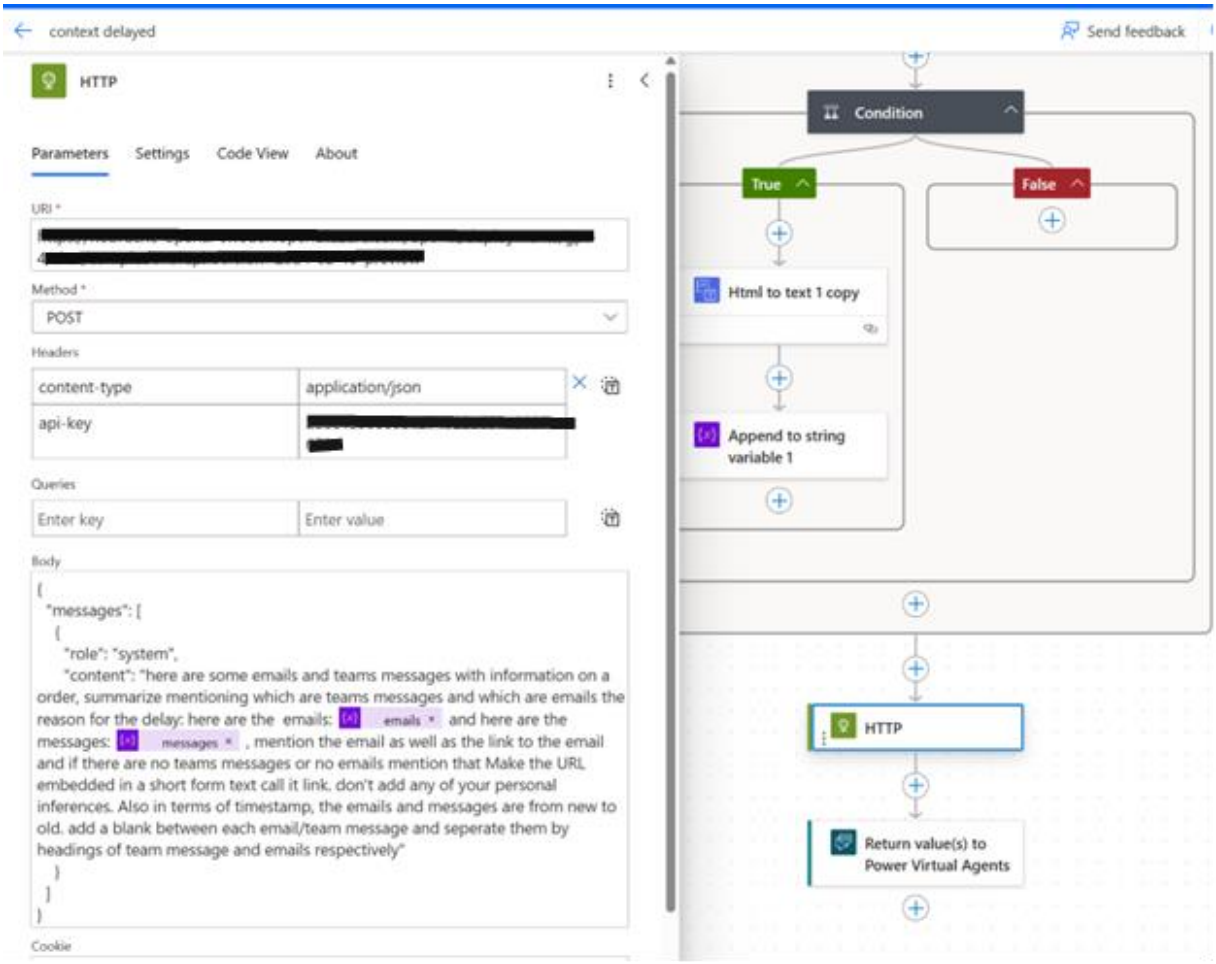


If it does, we will append the message as well as the WebUrl of the Teams message (this can be found from the dynamic variables).

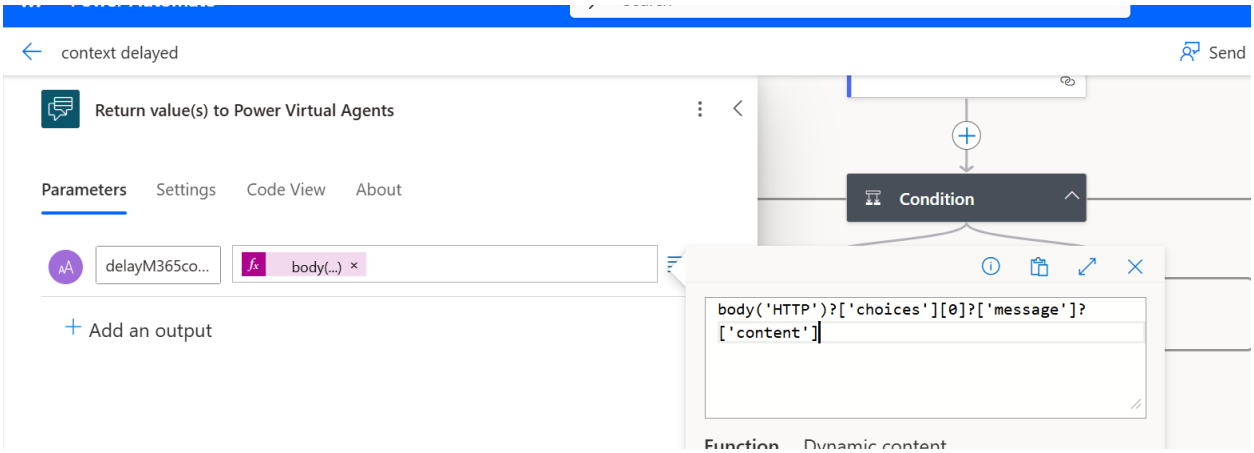


After the relevant Teams' messages and the URLs are saved in the variables, we will now send this as an input to Azure OpenAI to summarize and give the salesperson insights on why the order is late.

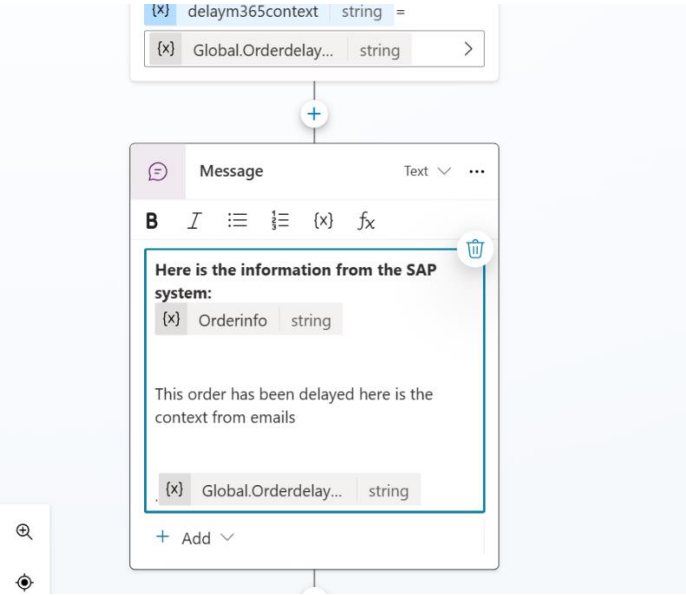
For this we will make an HTTP request to Azure OpenAI.



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

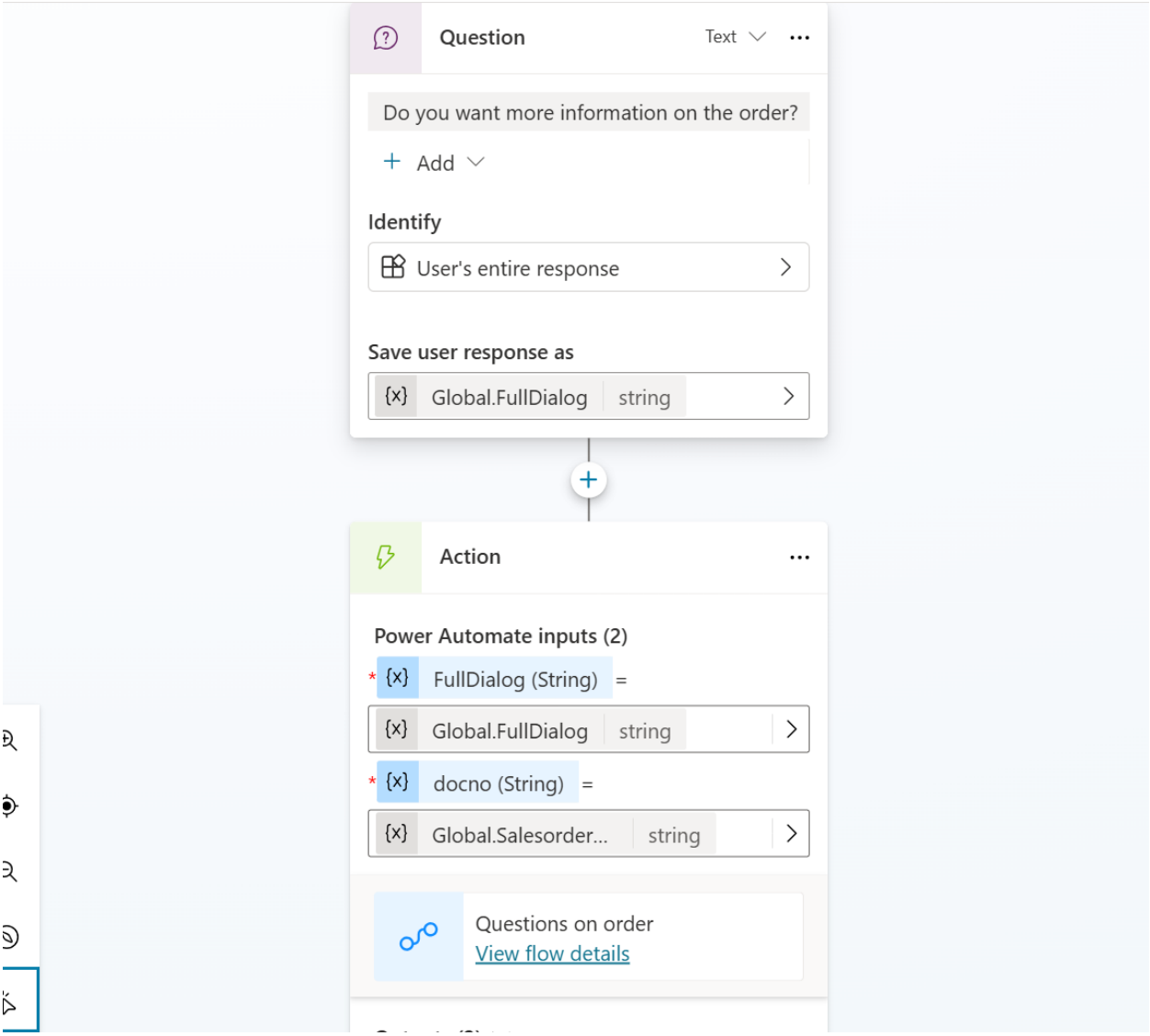


Finally, we will send the response back to Copilot Studio and present this information to the user as shown – the variable “orderinfo” is the output from the “get order status” step.



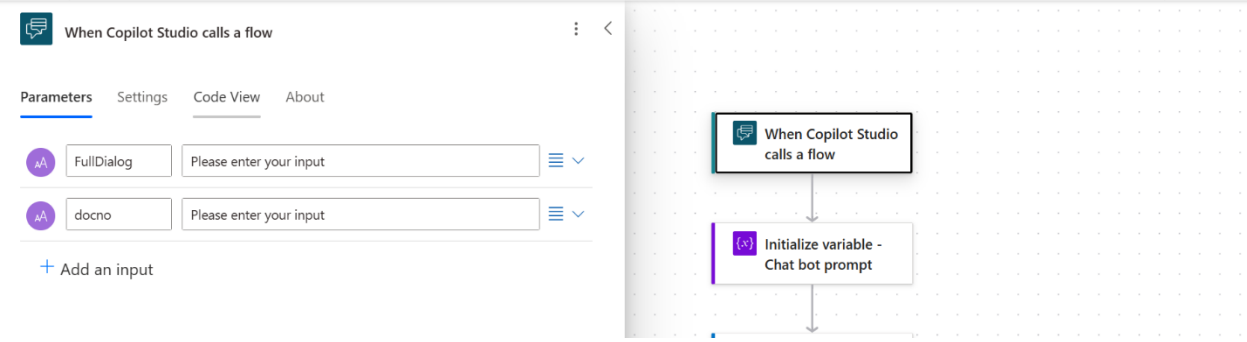
The next step after giving the order context is to allow the user to ask the bot any question they have about the order on the SAP data from the system.

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.

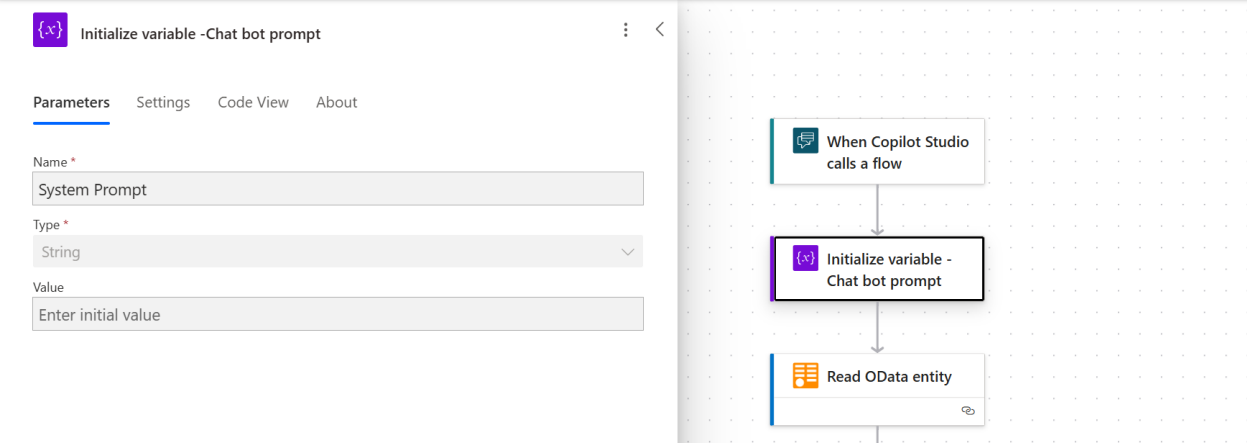


Here is the Power Automate flow for this, the name of this flow is “Questions on order - OData”:

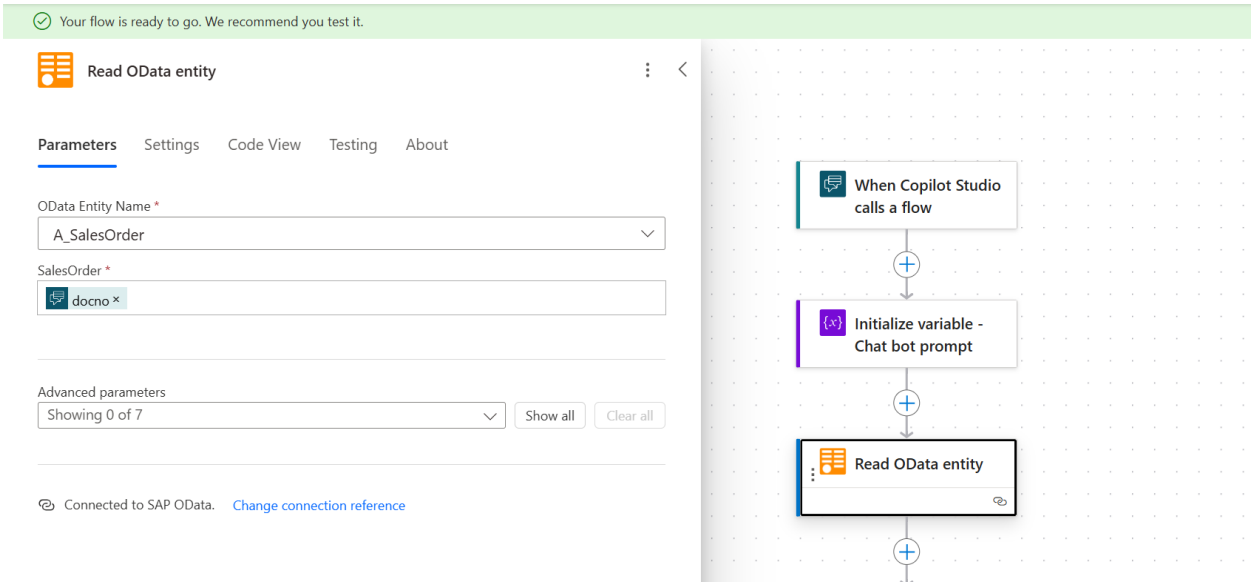
The first step is to add two inputs to this, the document number, and the question that the user is asking.



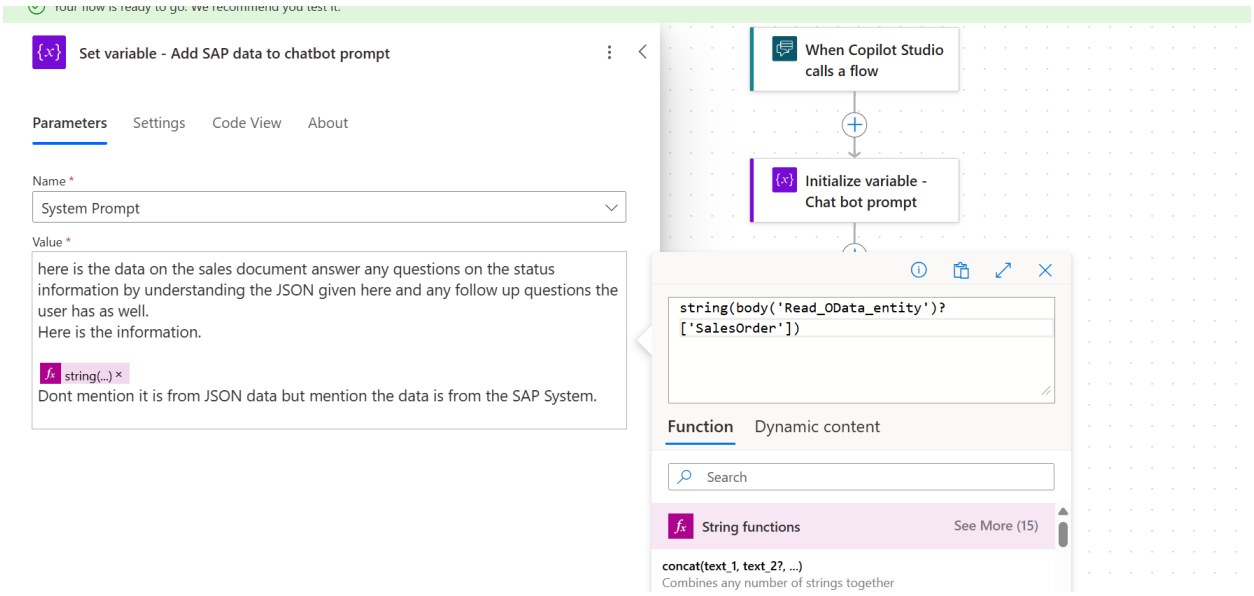
Next, we will initialize the Variable System Prompt that has to Azure OpenAI



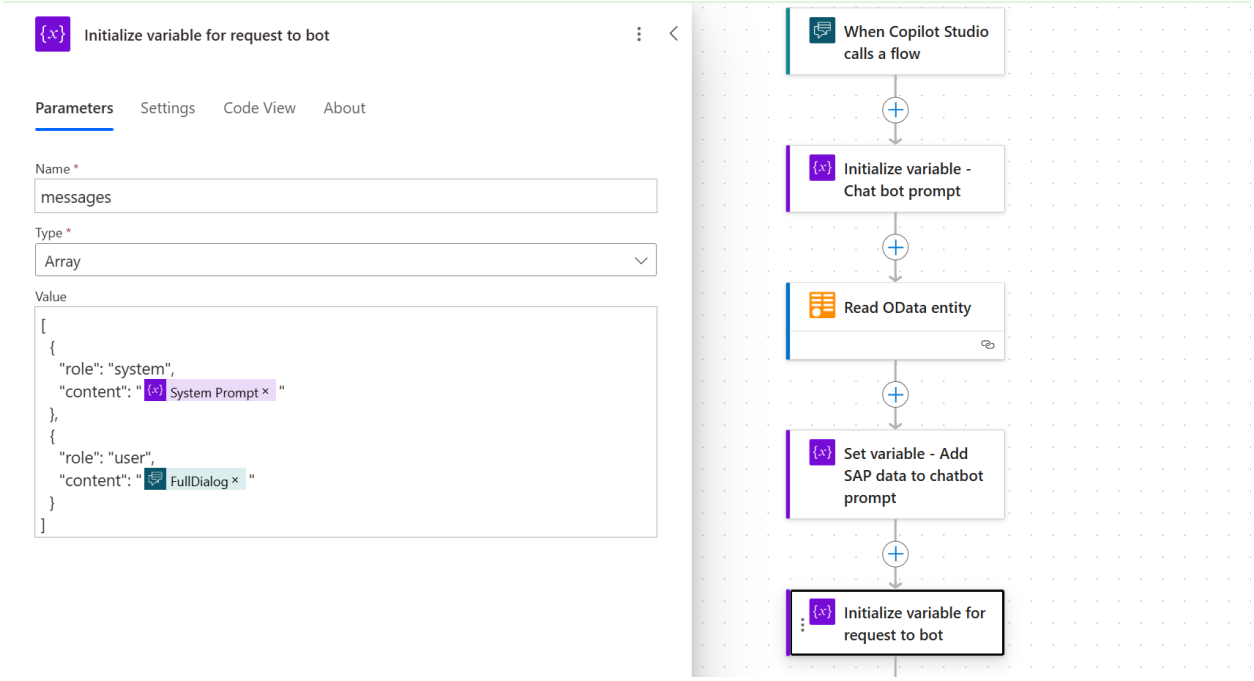
We will next once again use the Read Entity SAP OData connector with the Entra ID login (as done before), use the APIM URL for API_SALESORDER_SRV and input the document number sent from the Copilot Studio for the “Sales Order” variable.



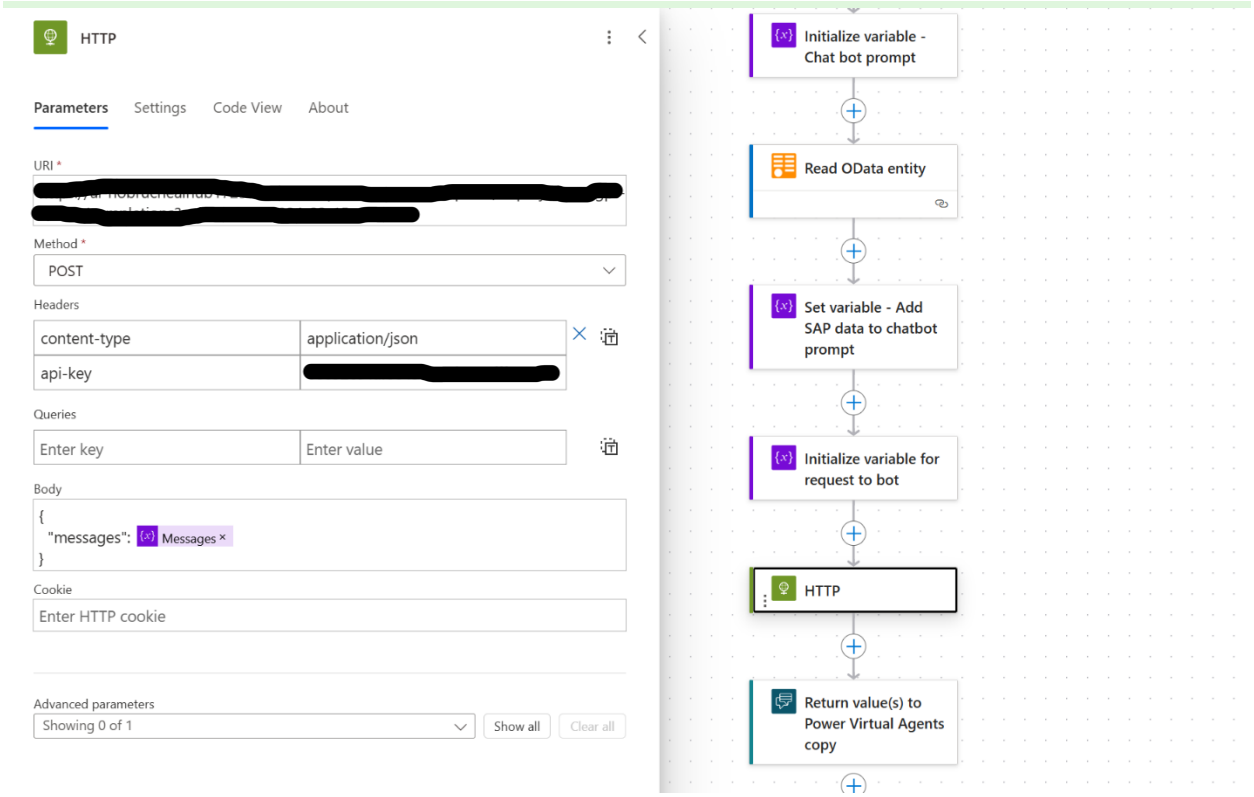
Once the data is fetched from the SAP system, we will now add the information about the sales order to the system prompt along to give the AI context on the order so that it can answer any question the user has about it. You can use the prompt below.



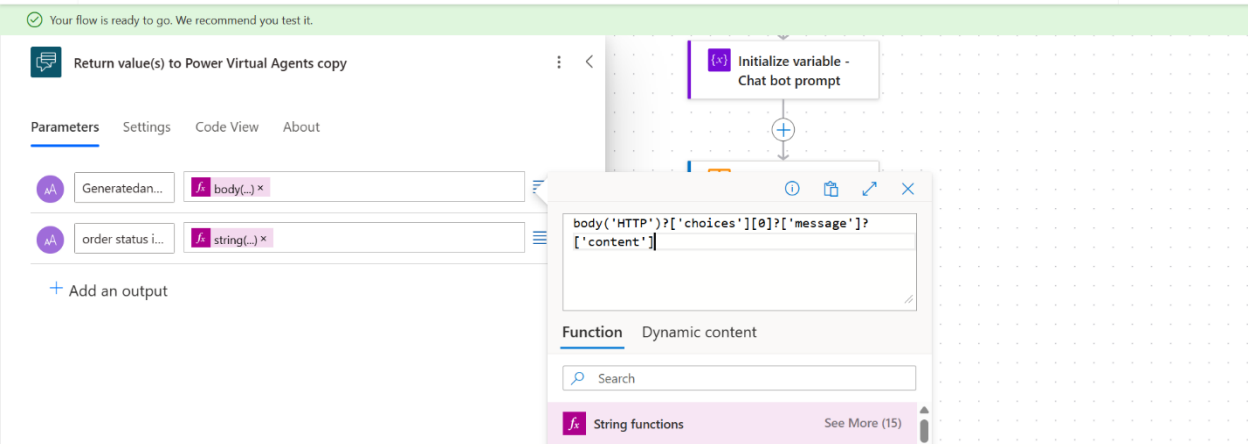
Next, we will add the question the user wanted to ask the AI under the “FullDialog” input variable to the request to be sent to the AI for it to answer the question.

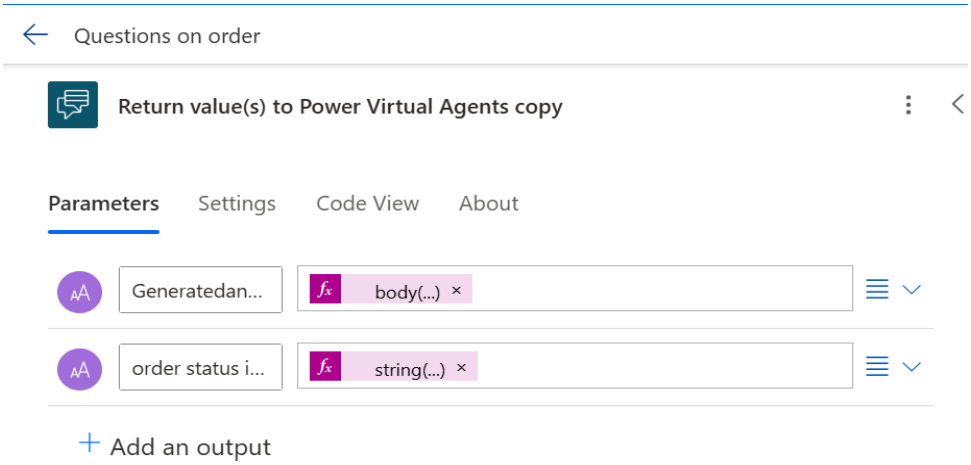


Lastly, we will now send the 'messages' variable to Azure OpenAI using an HTTP call as shown below:

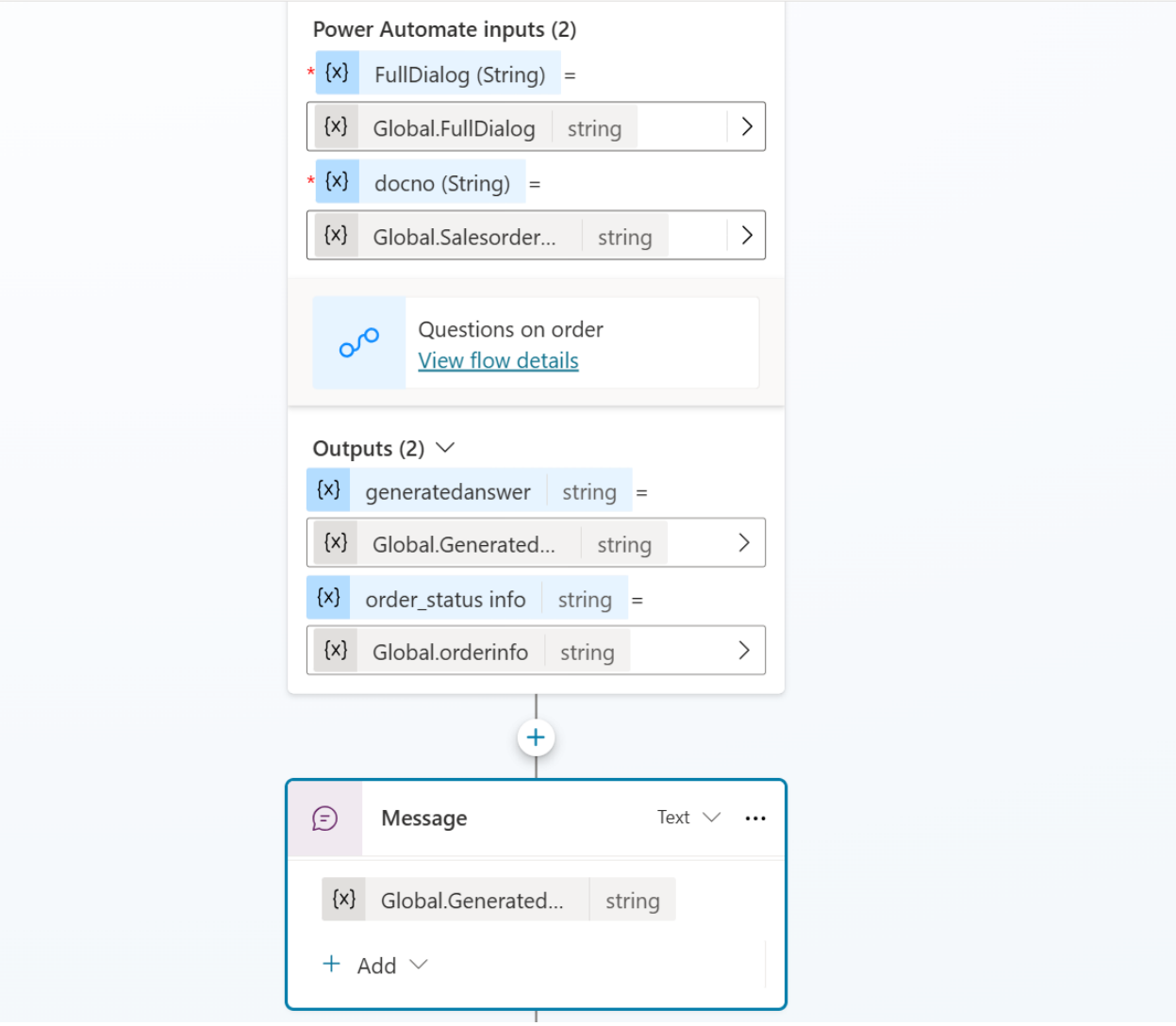


Lastly, we will process the response, store it in a variable, and send it back to the Copilot Studio – this variable will be called “Generatedanswer”. (Note: the value for the response is the same formula as the previous Power Automate flow: body('HTTP')?['choices'][0]?['message']?['content']).



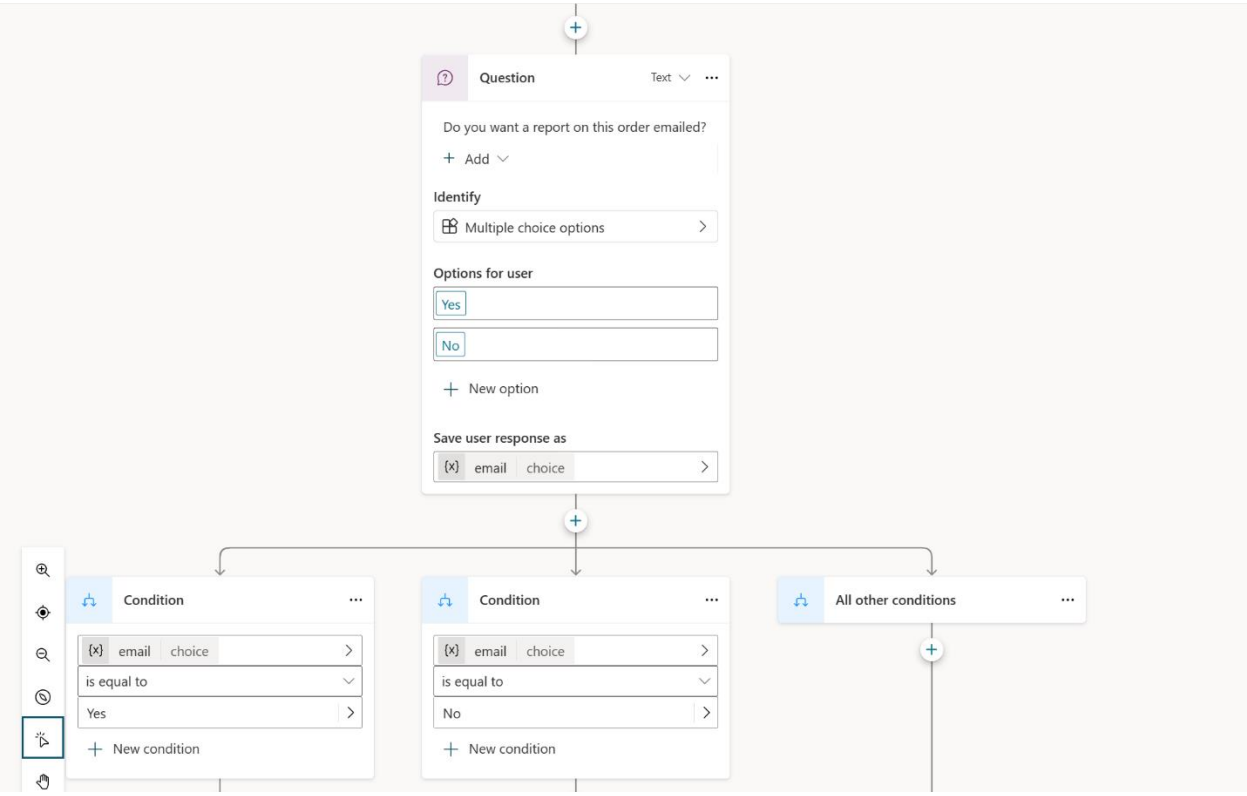


The other variable we will send back is the order status information we got from the SAP system. This variable could be used by other potential topics, and we save it as a global variable in the Copilot Studio so that it can be used across topics.

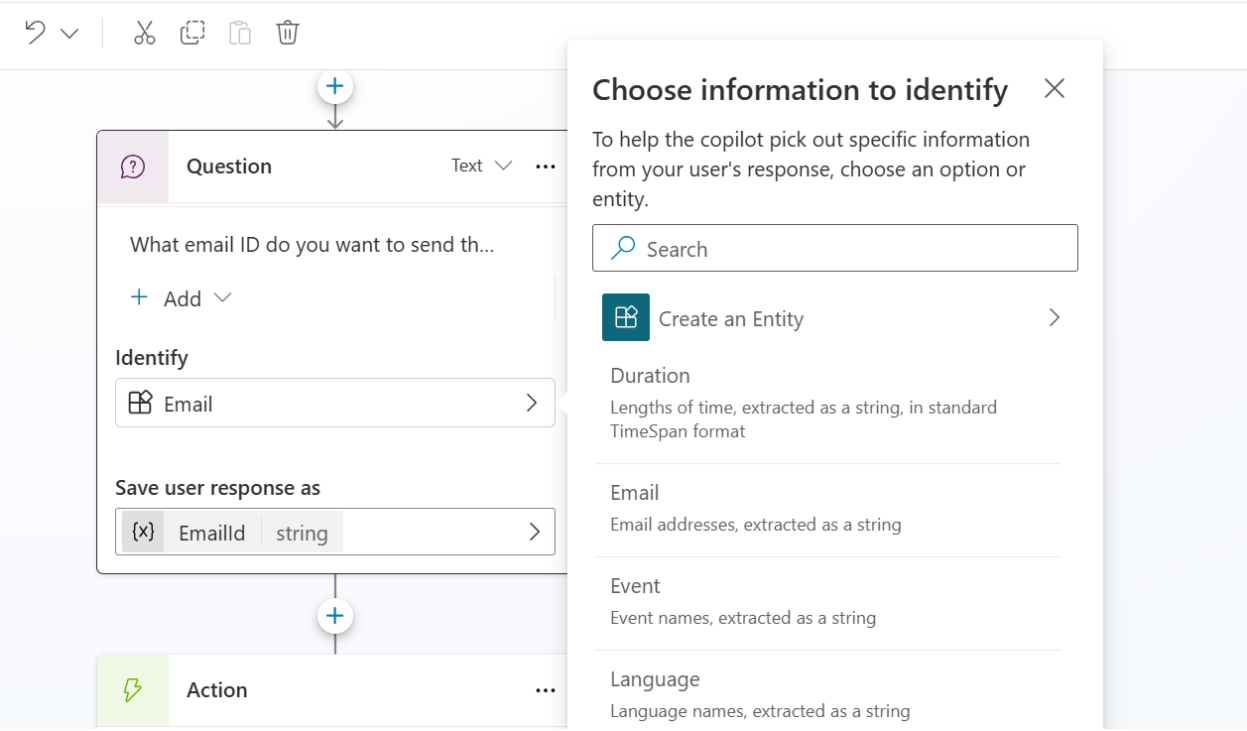


Lastly, if the salesperson wants to get back to the customer with the analysis, we will ask the salesperson if they want the entire conversation/ questions they asked about the order to be summarized by AI and put into a report to be sent to the customer via email on Outlook.

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

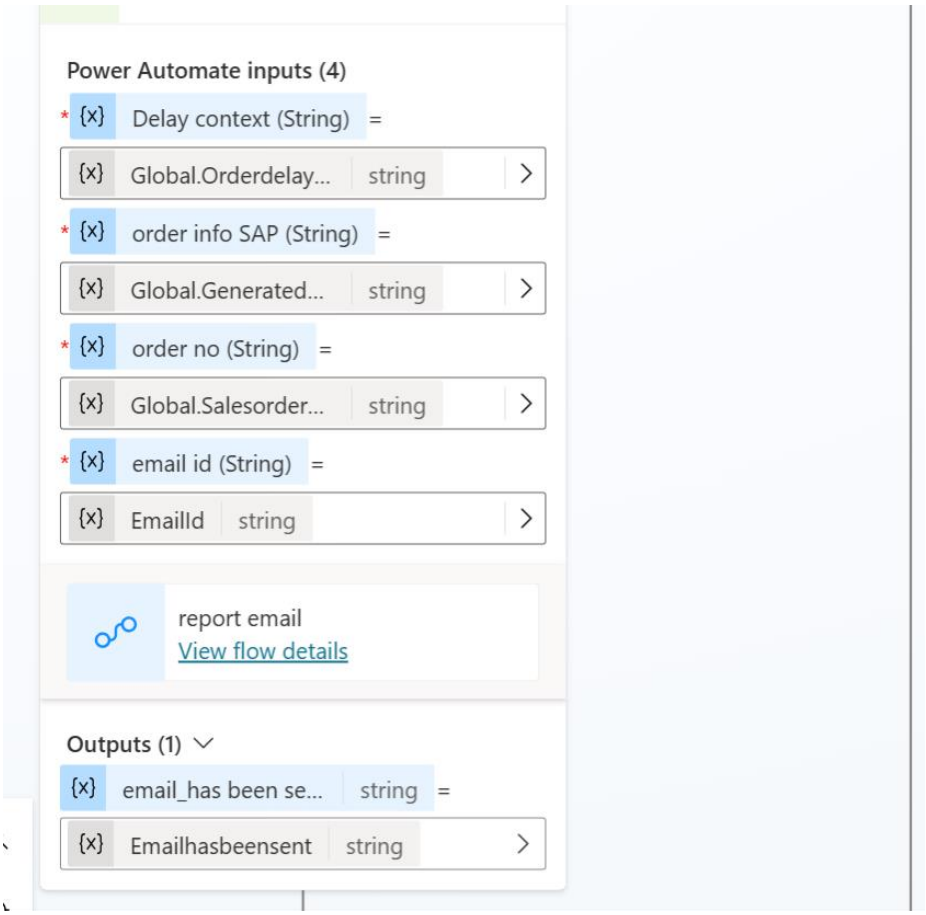


If the user responds that they want to send an email, we will ask them for the email ID they want to send this to, for this, we need to ask another question and select the ready-made email entity as the response (as we did for customer number and sales order number earlier).



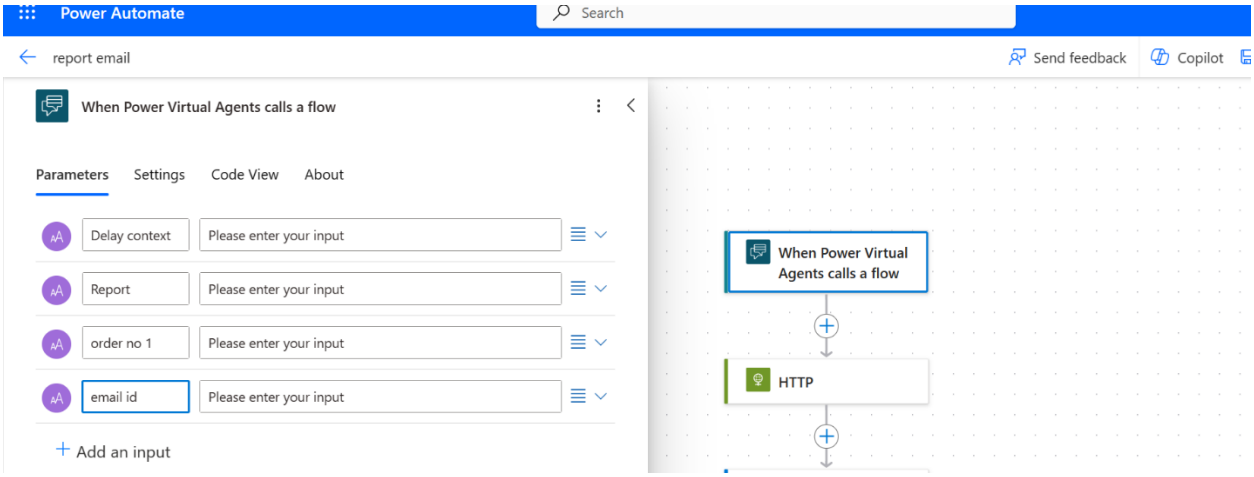
Finally, after that, we will call the final Power Automate flow with the variables shown below. This Power Automate flow is called “report email”.

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.

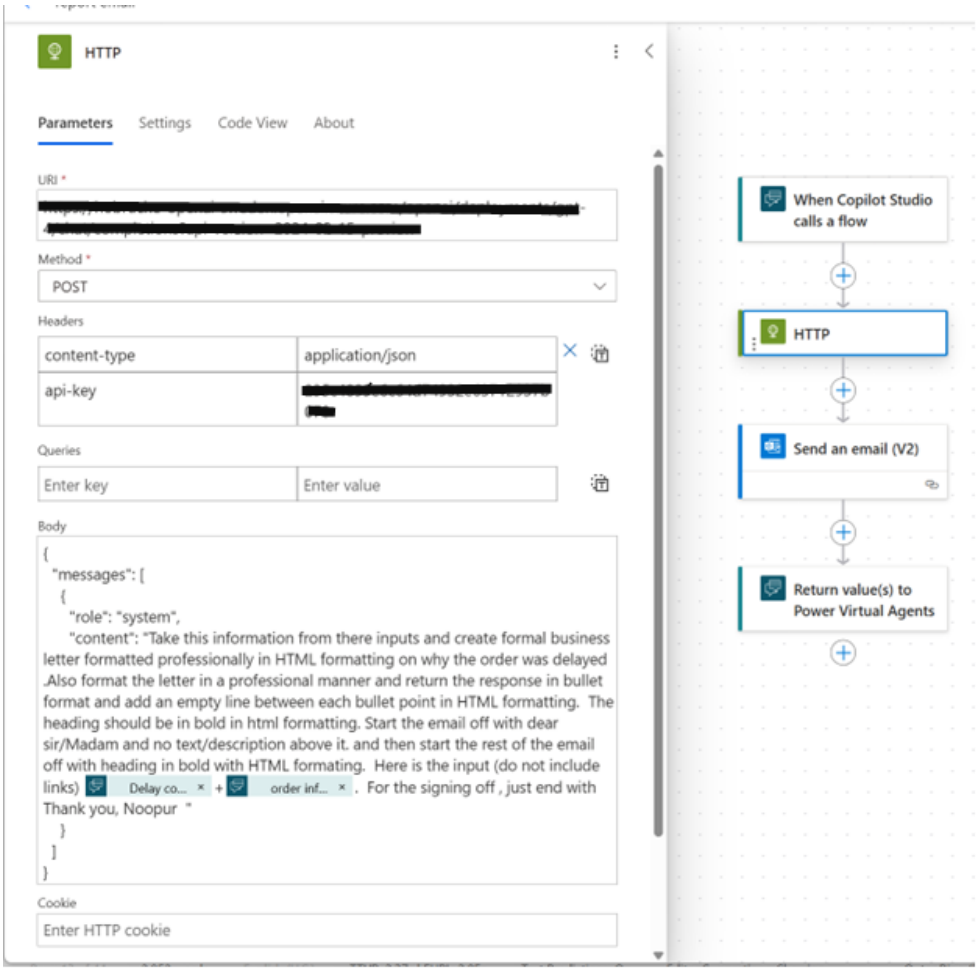


The steps for this flow are:

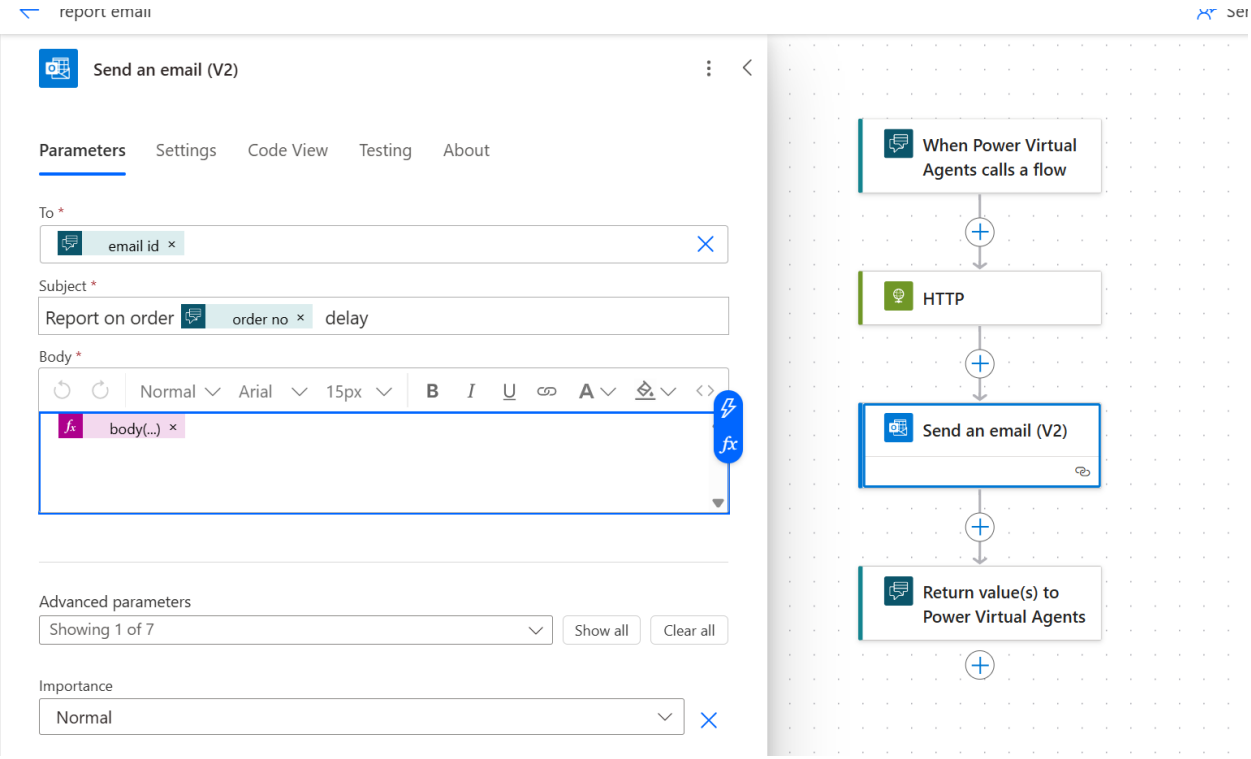
The first step is to define the 4 variables we need. The summary of Outlook + Teams messages, the response to the previous question asked from Azure OpenAI, the order number and finally the email ID we want to send the email to – this would be the customer's email ID.



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:

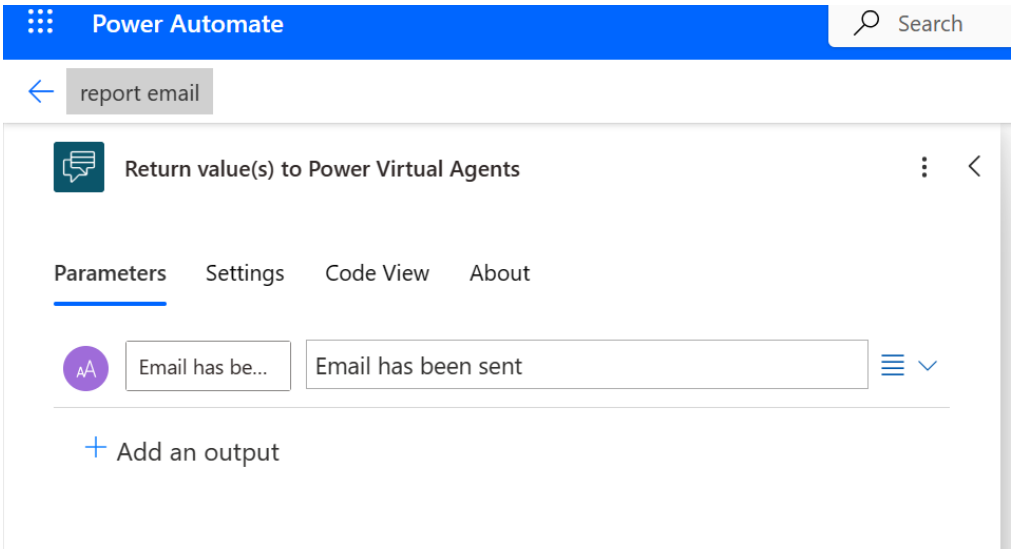


Next, we use the outlook send email function to send the email to the customer and populate the fields as shown below.

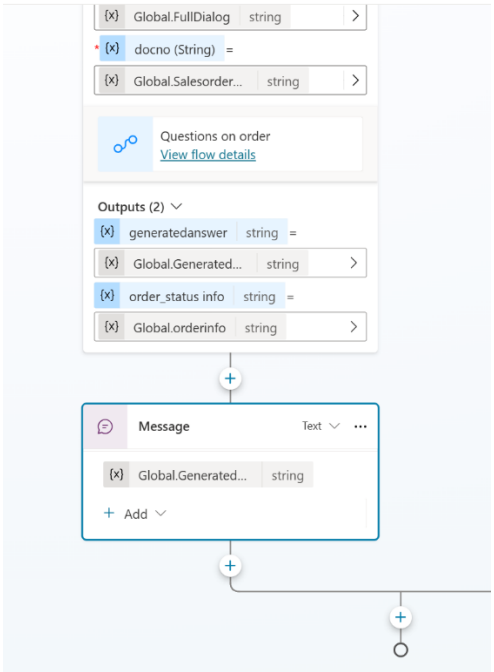


The body of the email here is the response from Azure OpenAI (which is the generated email in this case) - this is the expression for it as used before body "('HTTP')?['choices'][0]?['message']?['content']".

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



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



Steps to recreate the Scenario part with Adaptive Cards and Material Recommendations.

Information returned via chatbots should not be restricted to looking only as good as the UI of the platform you deploy the bot on, it would more interactive and personal to your brand to have a way to choose the design in which users get to interact with the bot you created. This is where adaptive cards are a game changer. Adaptive cards allow you to create responses from the bot that are more visually appealing, interactive, and customizable and make a fantastic addition to your flow on Copilot Studio.

To explore how it could be used as shown in the demo video, go back to the Copilot Studio topic – 'Sales order statuses, and we will swap the text responses at 2 places for adaptive cards. (Or you could just read about how to create an adaptive card and use it elsewhere).

Here are the two places we will add the adaptive cards.

1) Getting the sales order number to deep dive into from the customer.

2) Displaying information about the order from the SAP system.

To start off, navigate to the node in which we ask the user for the sales order number they want to deep dive into.

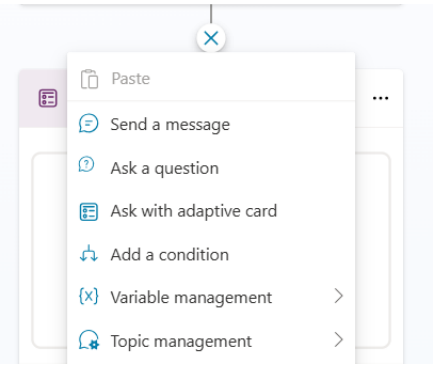
Instead of asking a question, we can replace it with an adaptive card.

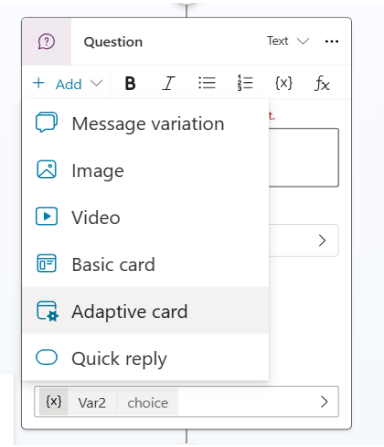
You can add an adaptive card like this:

1.Add a question node

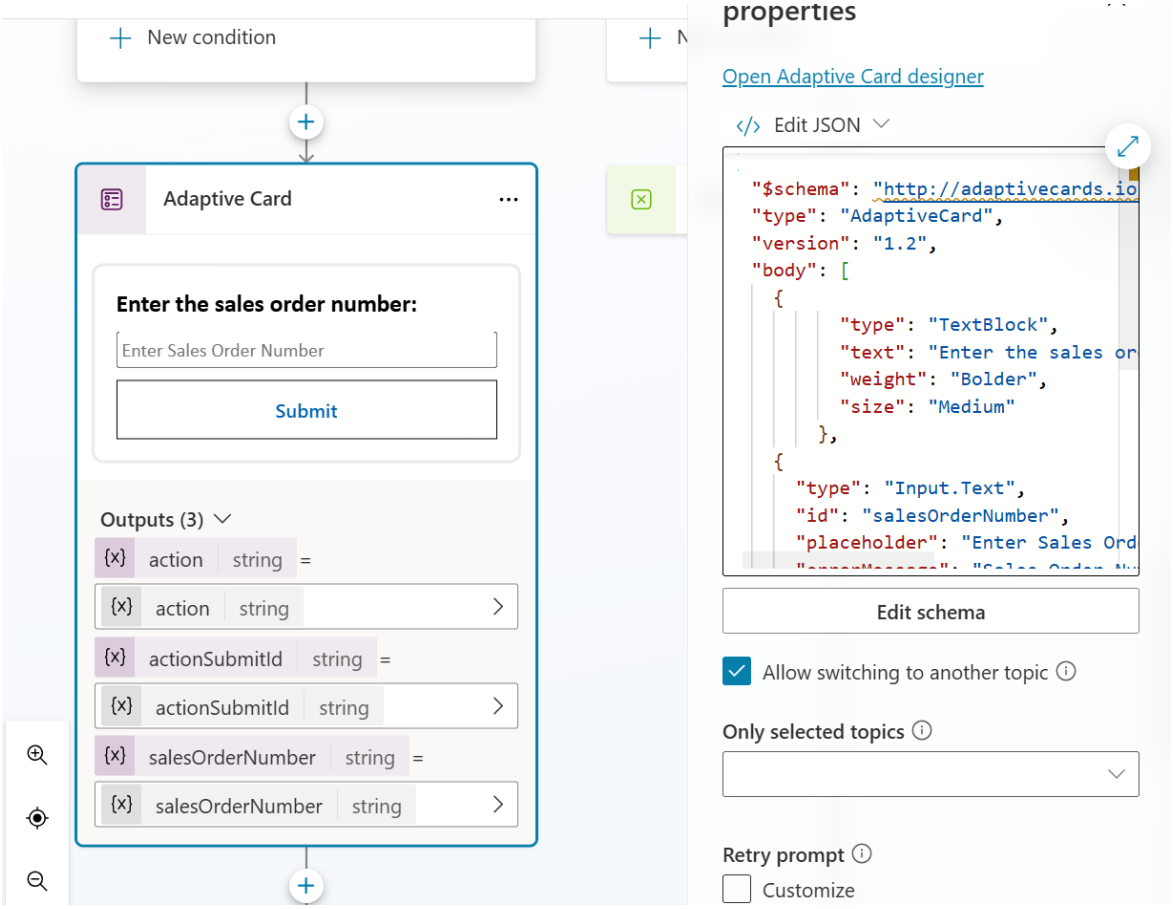
2. Click on add adaptive card.

3. Double click on the adaptive card to view/ modify the code.



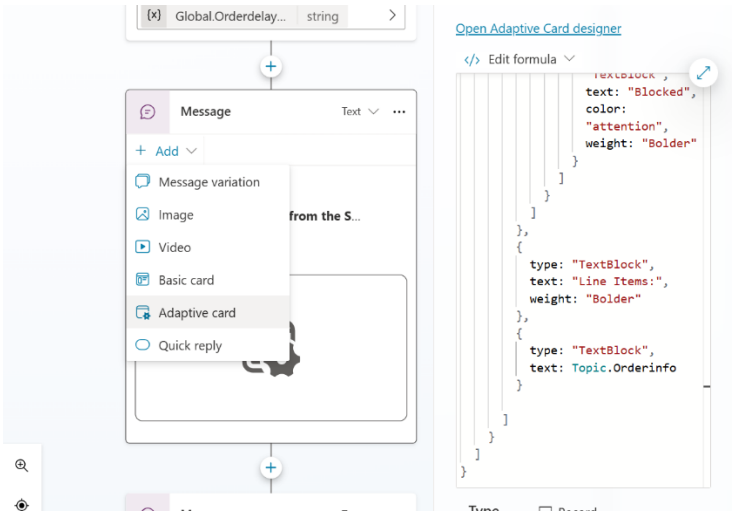


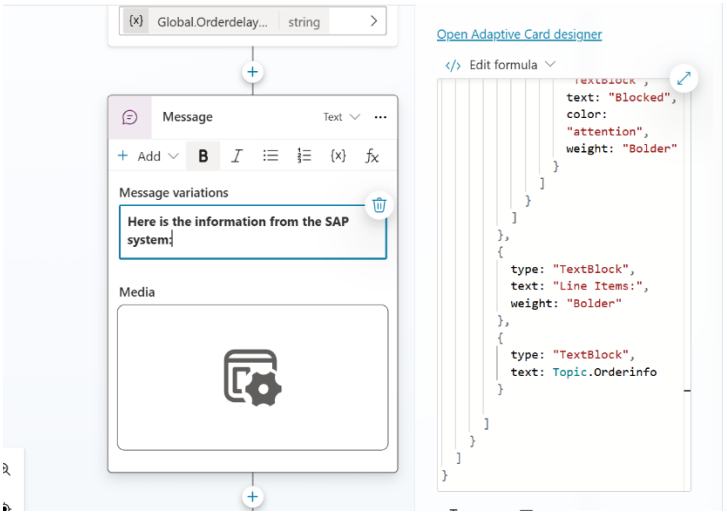
Here is the adaptive card we will use to get the sales order number from the user. (The code for this is available on GitHub) and you can copy paste It in the “edit JSON” text box shown below.



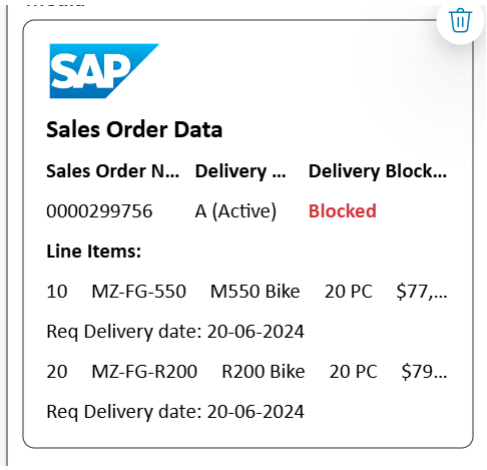
Now to display the information about the order from the SAP system, we will once again make use of an adaptive card to display the information in a more visually appealing way.

Navigate to the part in the Sales order flow after the Power automate flow “context delayed” is called and add an adaptive card to the message node. Click on add, and add an adaptive card to the message node.



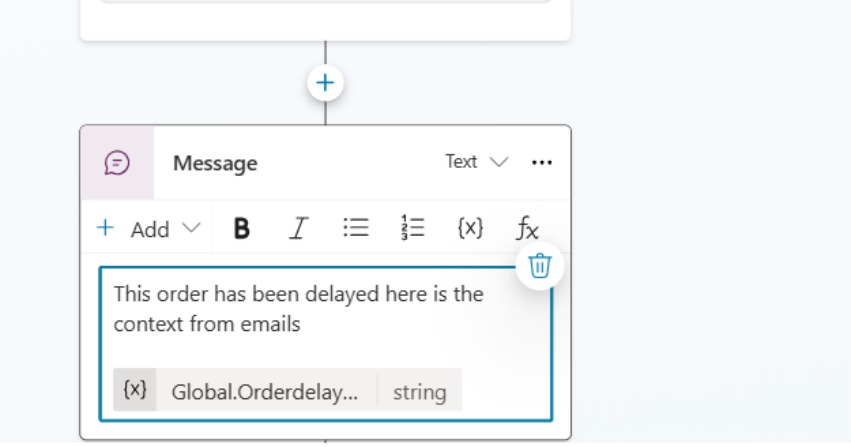


This is how we want the adaptive card to look while in the flow. You can copy the code of this adaptive card from GitHub from the file named “SalesorderviewAdaptivecard.json”.



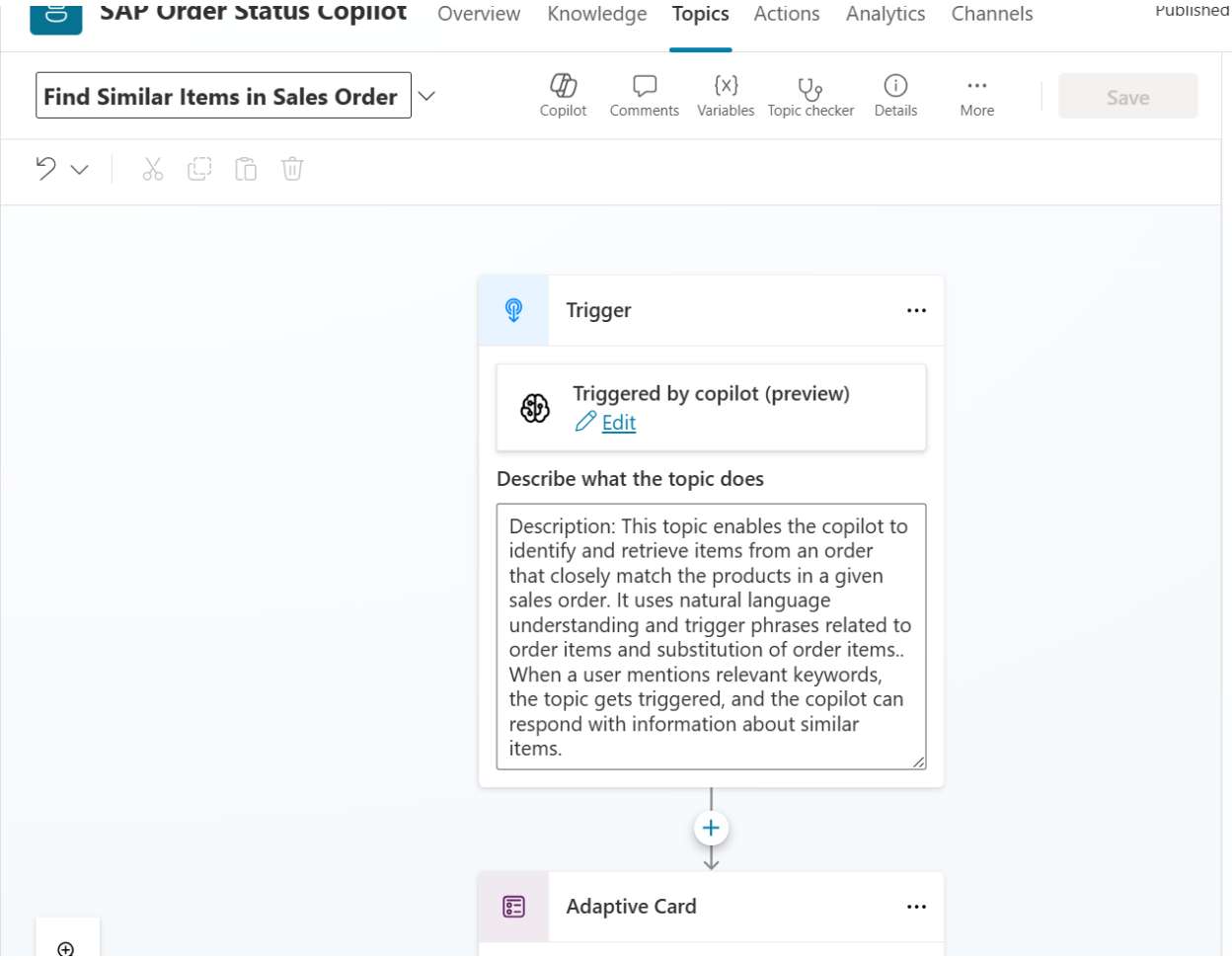
Note: For this adaptive card click on ‘Edit JSON’ and pick ‘Edit Formula’ instead as we will be entering dynamic values from the flow which are considered formulas.

Next, create a new message and add the variable with the MS365 context there.

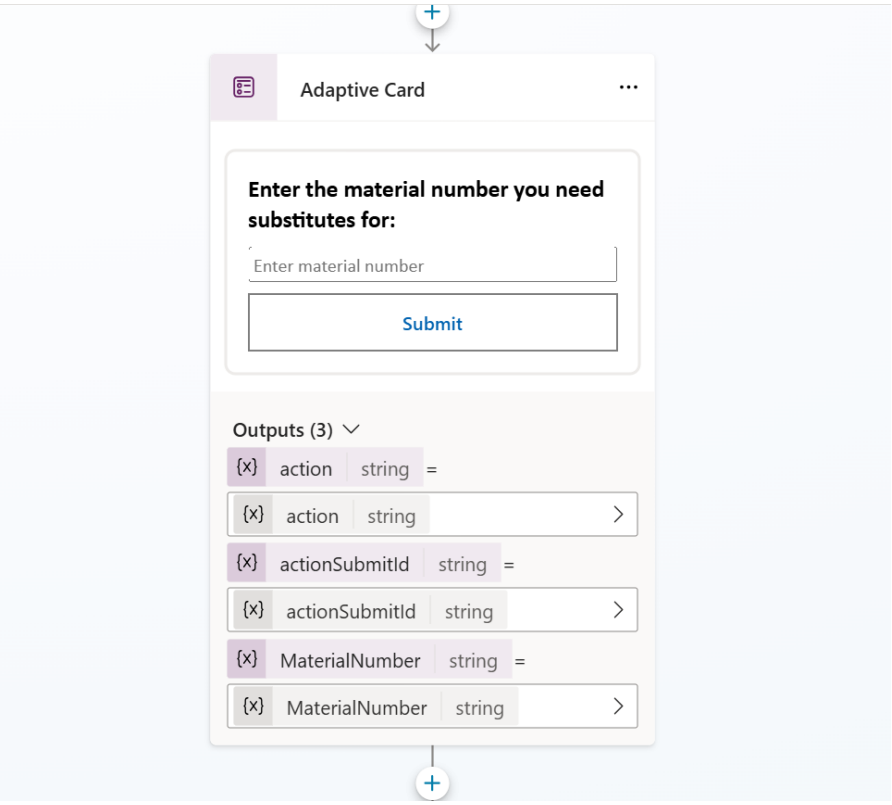


Now once, the salesperson has realized that one of the materials in not available (as shown in the demo video) we will create the topic that helps find a replacement item for material.

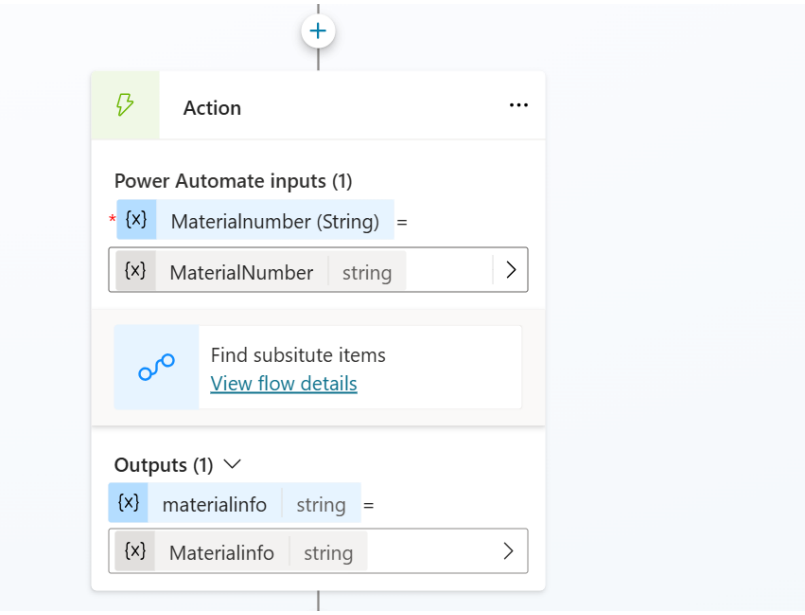
This topic will be called “Find similar items in Sales order” and the description will be as shown below.



Next, we will create an adaptive card to ask the user what sales order they want to find the item for. The code for this adaptive card will be on GitHub under the name ‘SalesordersubsituteAdaptivecardOdata.json’.

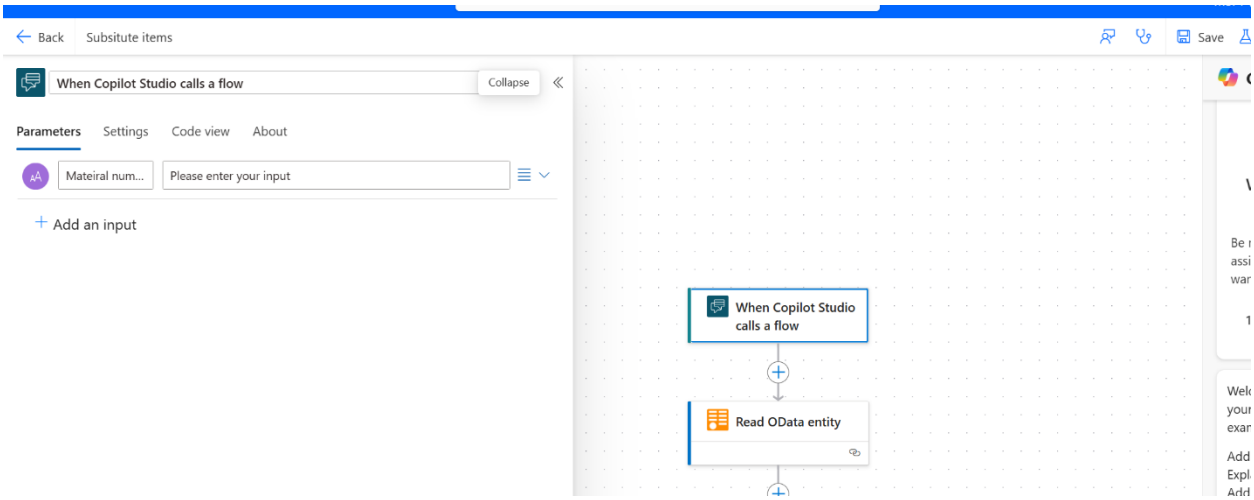


Now, we will call a Power Automate flow with the material number of the we want to find replacement for as the input.

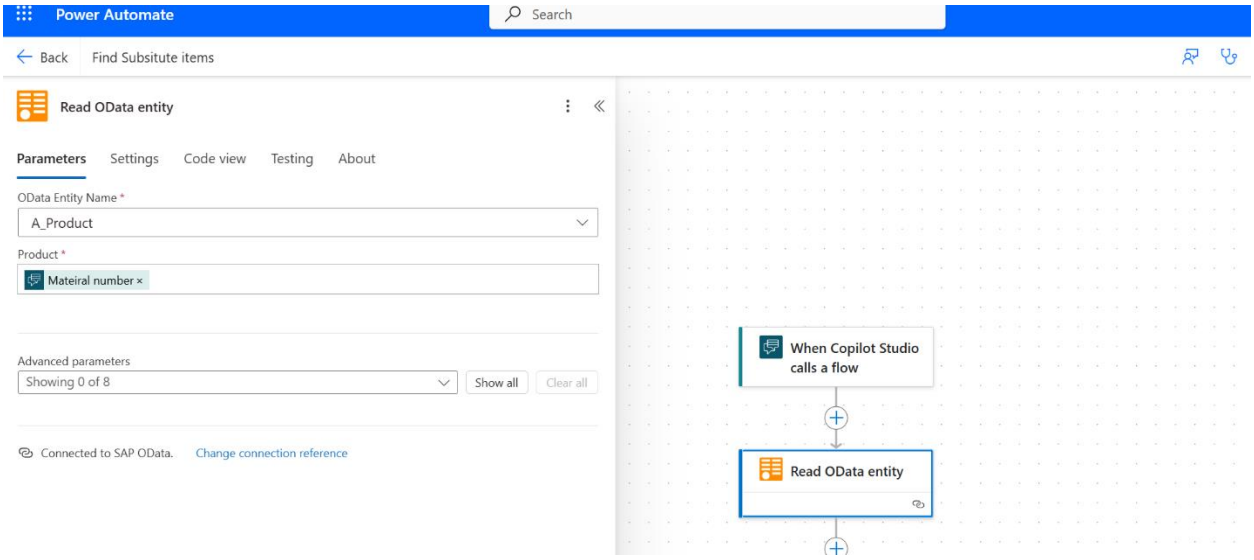


This Power Automate flow is called “Find item substitutes- OData”.

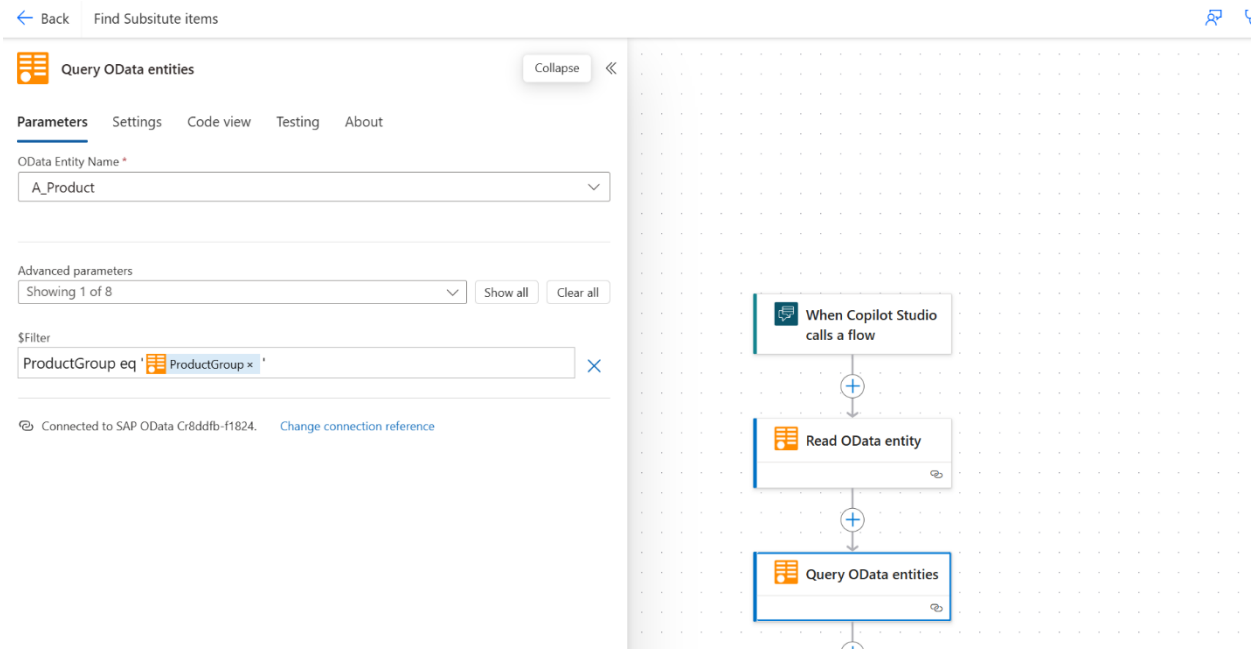
The first step to find the substitute items for an order is to get a list of all materials from the SAP system, for this we will call use the OData Service – API_PRODUCT_SRV which has all material related information.



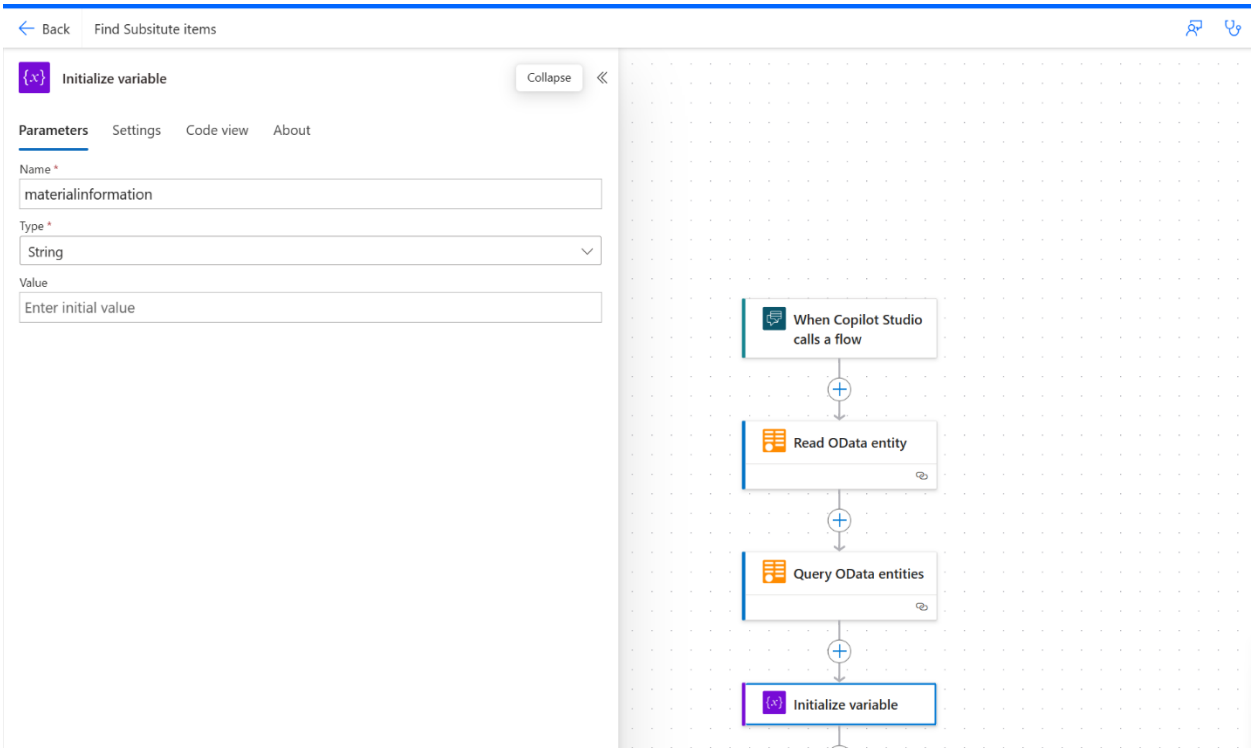
The first step will be to read the Material information for the material to be replaced and find out which product group it belongs to.



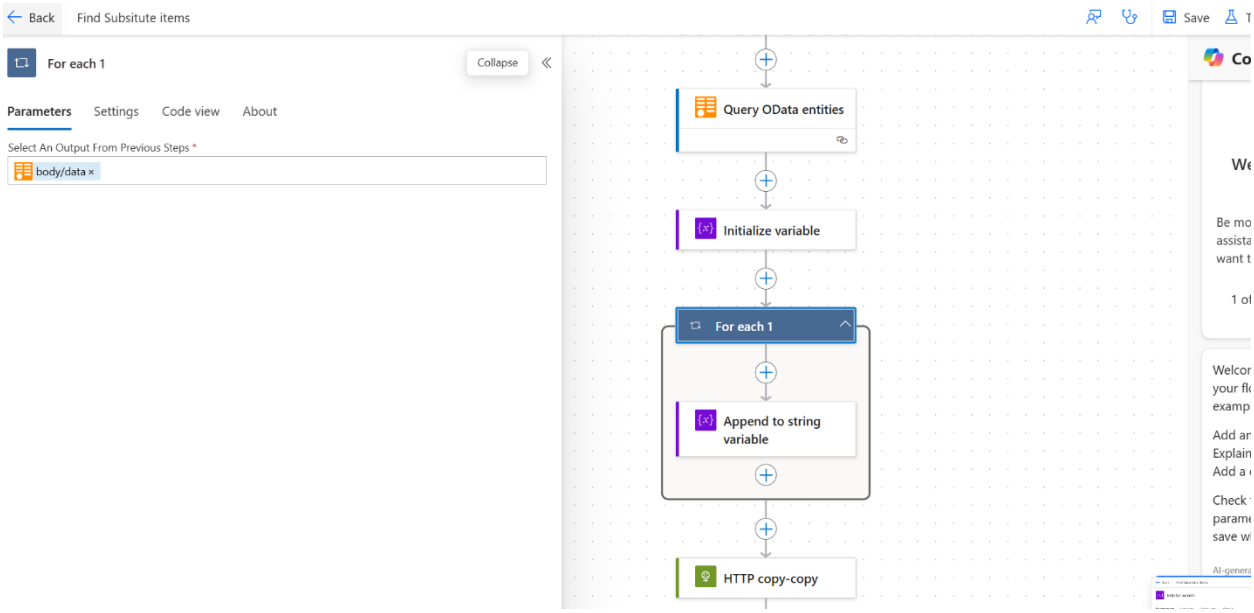
Next, we will query all the products from the system with the same product group.



We will then initialize a variable called ‘materialinformation’ to store the information about the new materials.

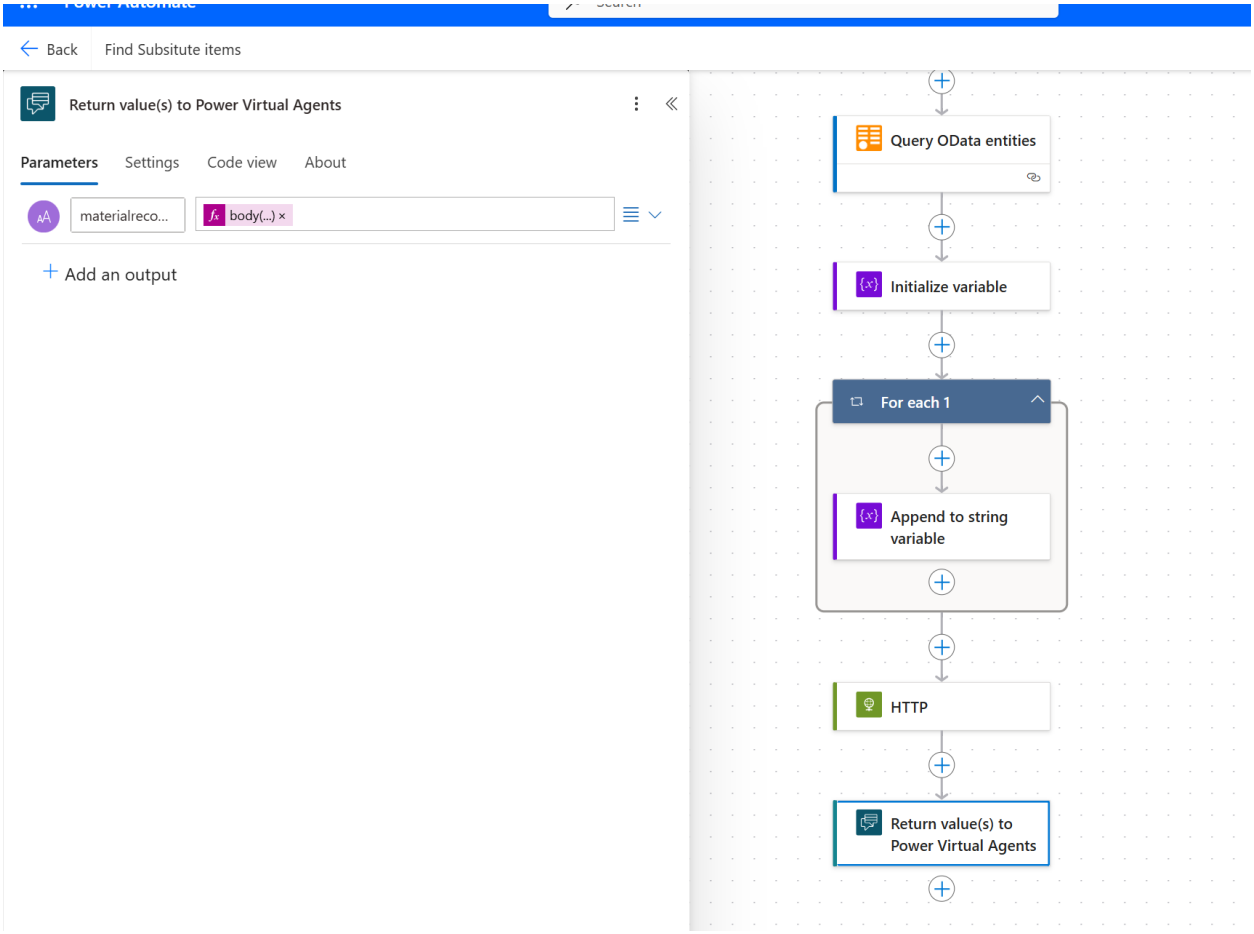


Then we will loop over each material returned from the Query OData action and append each item to the ‘materialinformation’ variable.

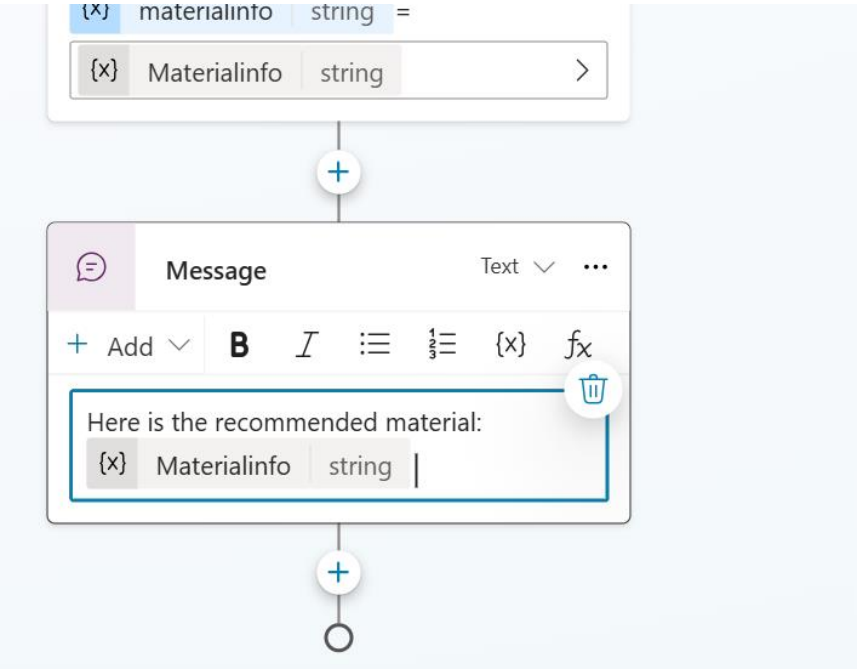


Finally, we will give the AI the prompt as shown below, the ‘Body’ or information from the material we need to replace, and a list of all the alternative materials stored in the variable ‘material information’.

Then we will return the response from Azure OpenAI back to Copilot studio. With the content being the function: body('HTTP')?['choices'][0]?['message']?['content'])

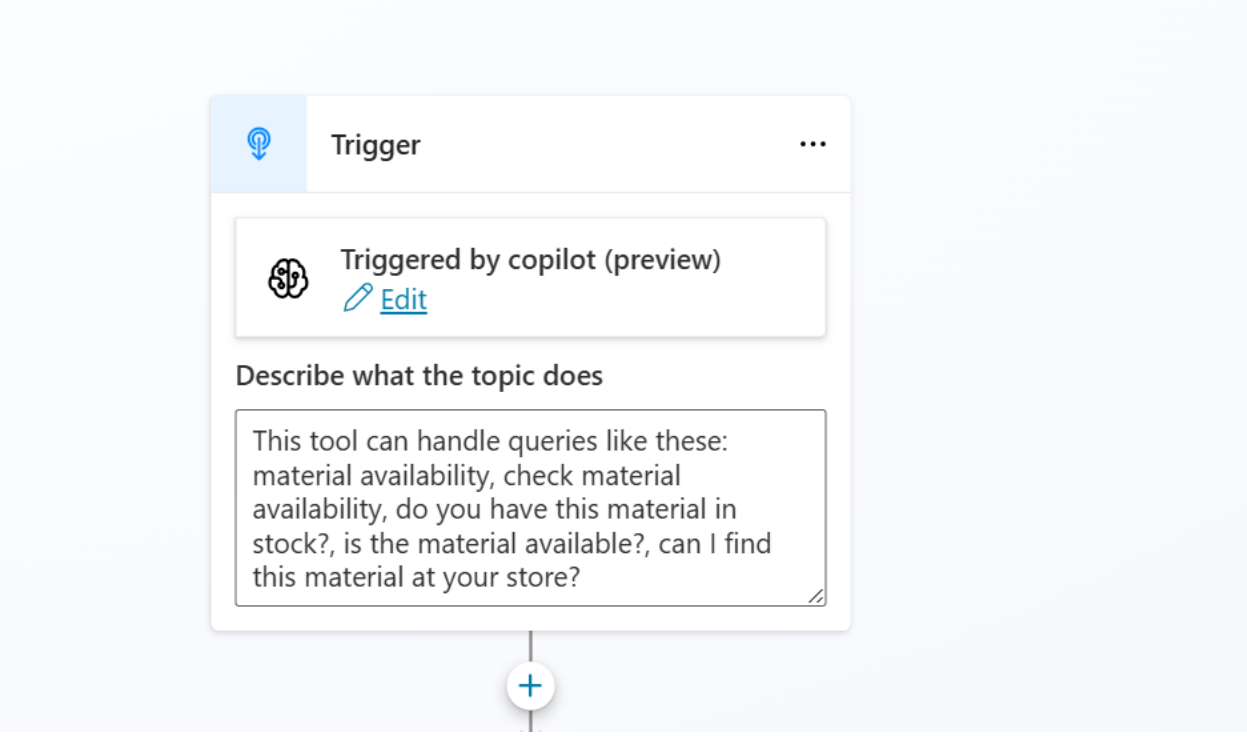


Back in the Copilot Studio we will return the output from the Power Automate flow back to the user as a recommendation.

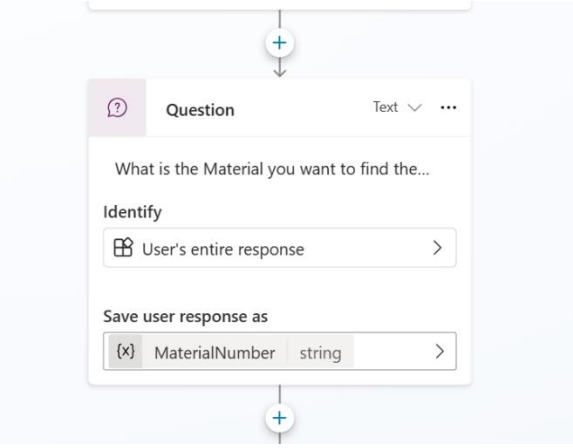


Once a substitute item has been selected, the user now wants to check the material availability of the recommended item.

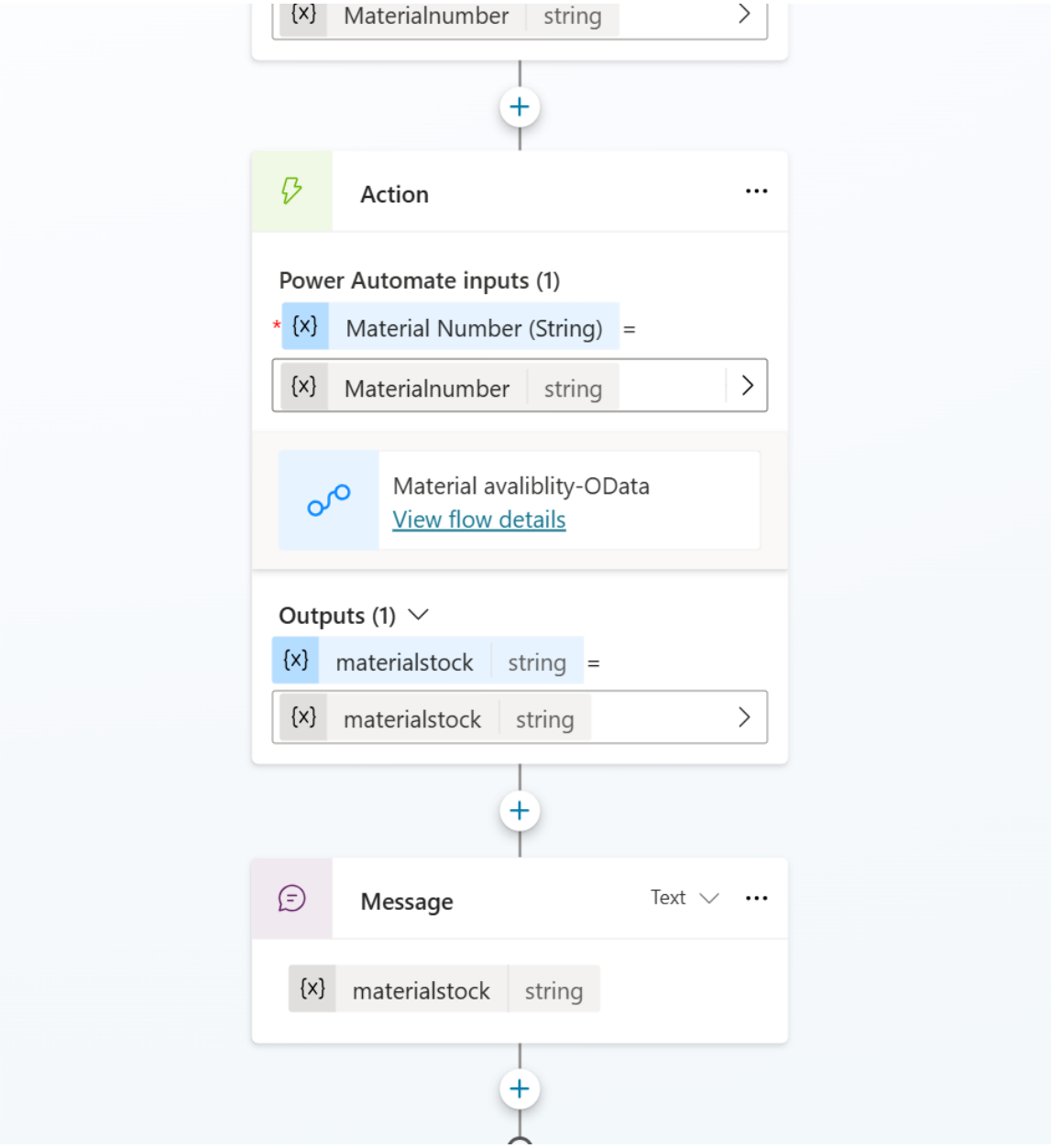
First Create a Copilot studio topic called “Material Stock information” with description as shown below:



Now we will ask the user to enter the material number they need the information for, for this we will once again make use of entities like we did in the previous steps and as explained here.

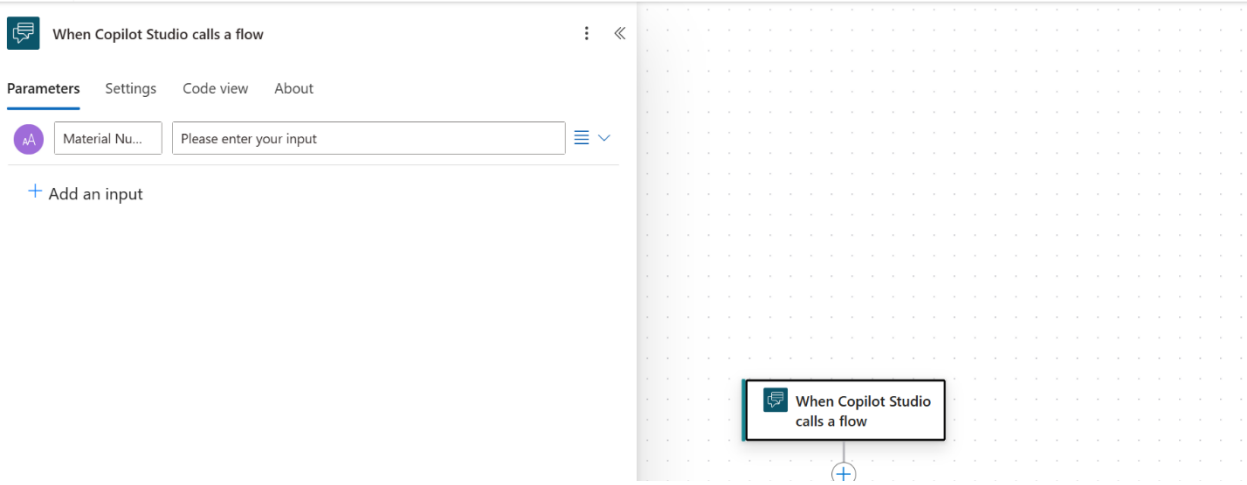


Now we will send this material number to a Power Automate flow which is called “Material Availability-OData”.

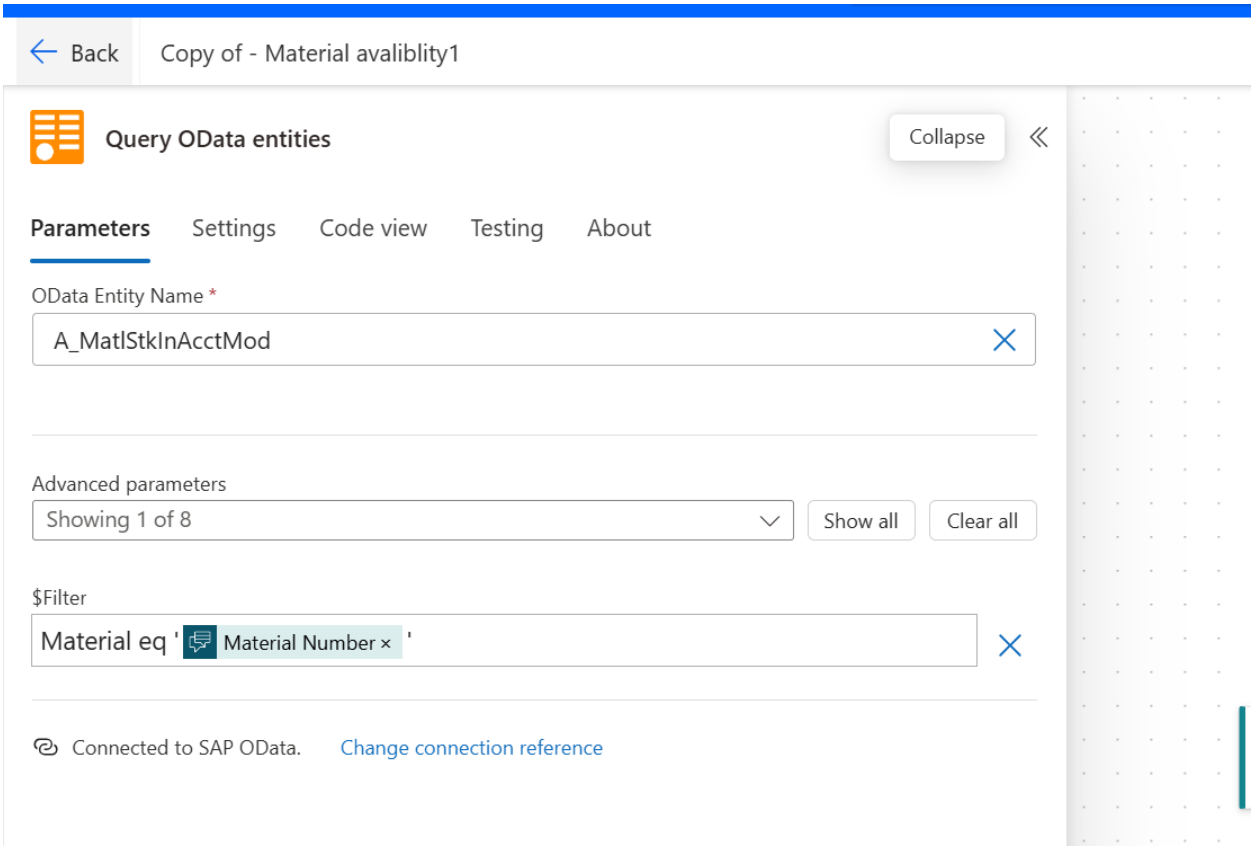


Here are the steps of the Power Automate flow called ‘Material Availability – OData’

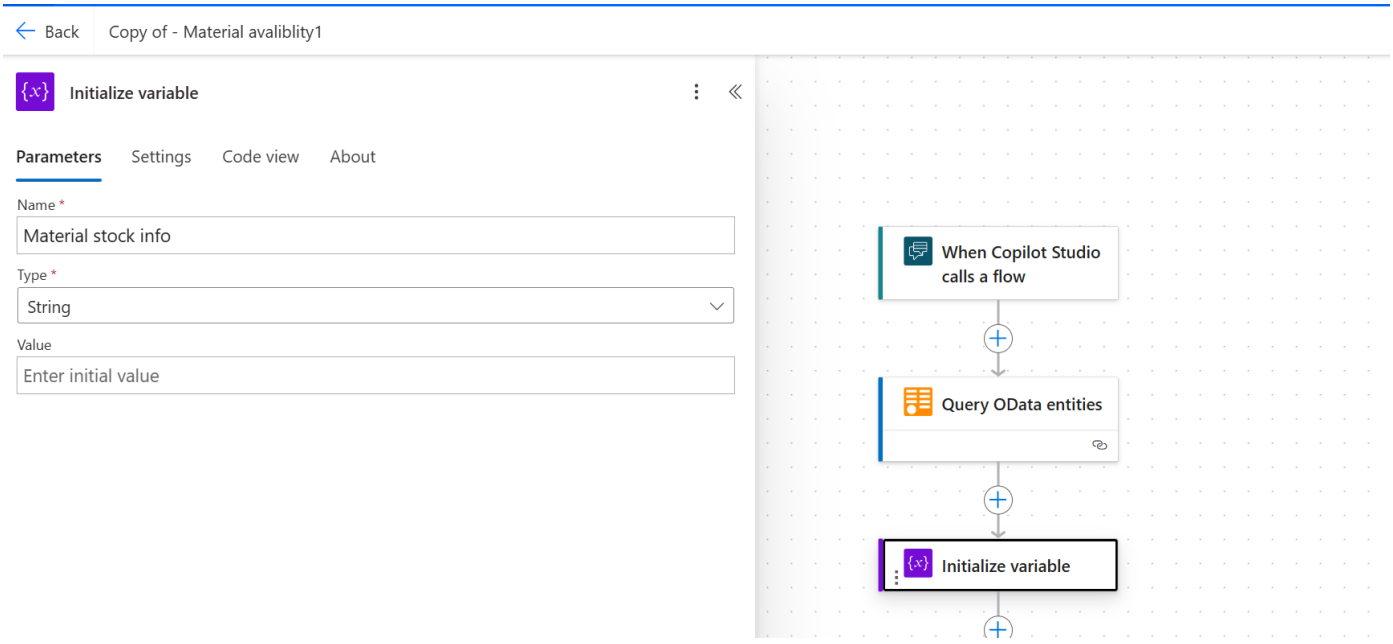
First, the input is the material number the user needs availability for from the Copilot Studio.



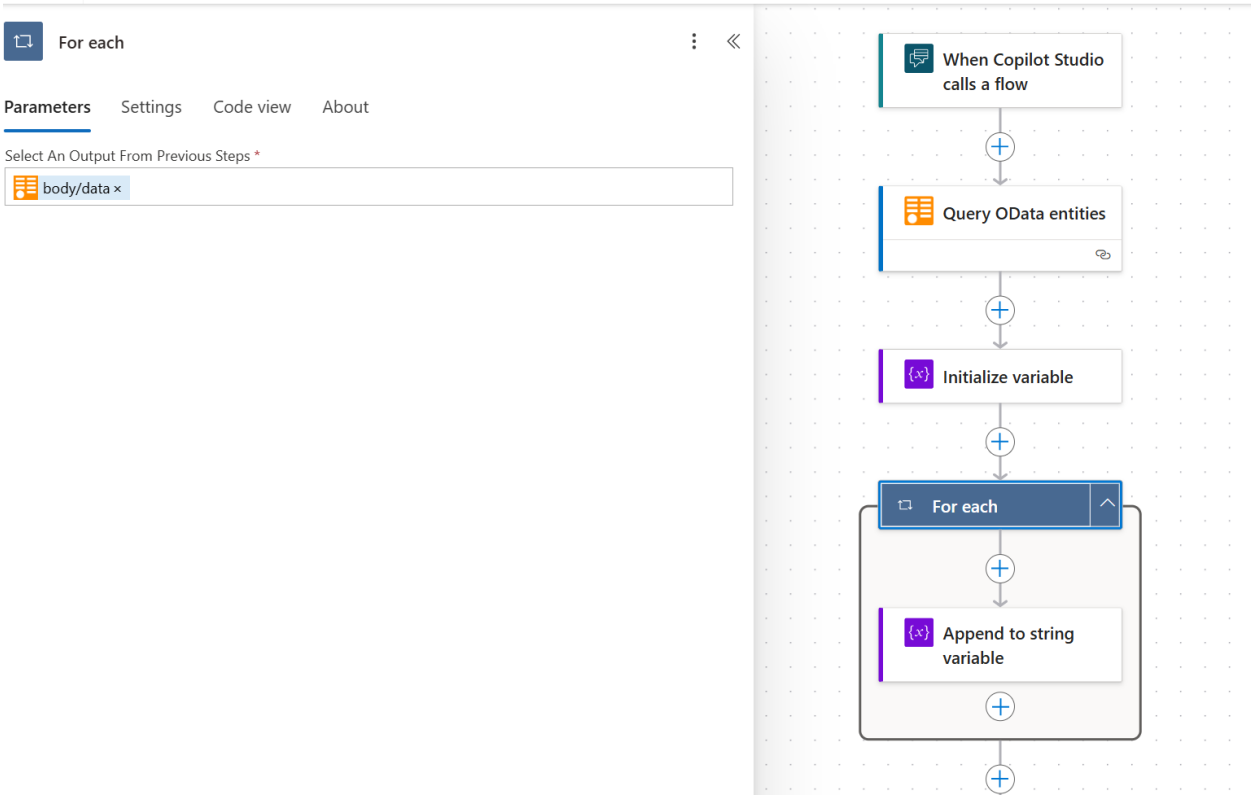
Next we use the SAP OData Query Connector option and create an APIM API created for API_MATERIALSTOCK_SRV and select the entity A_MatlStklnAccMod and add the Filter as shown below. This will give information about the material.



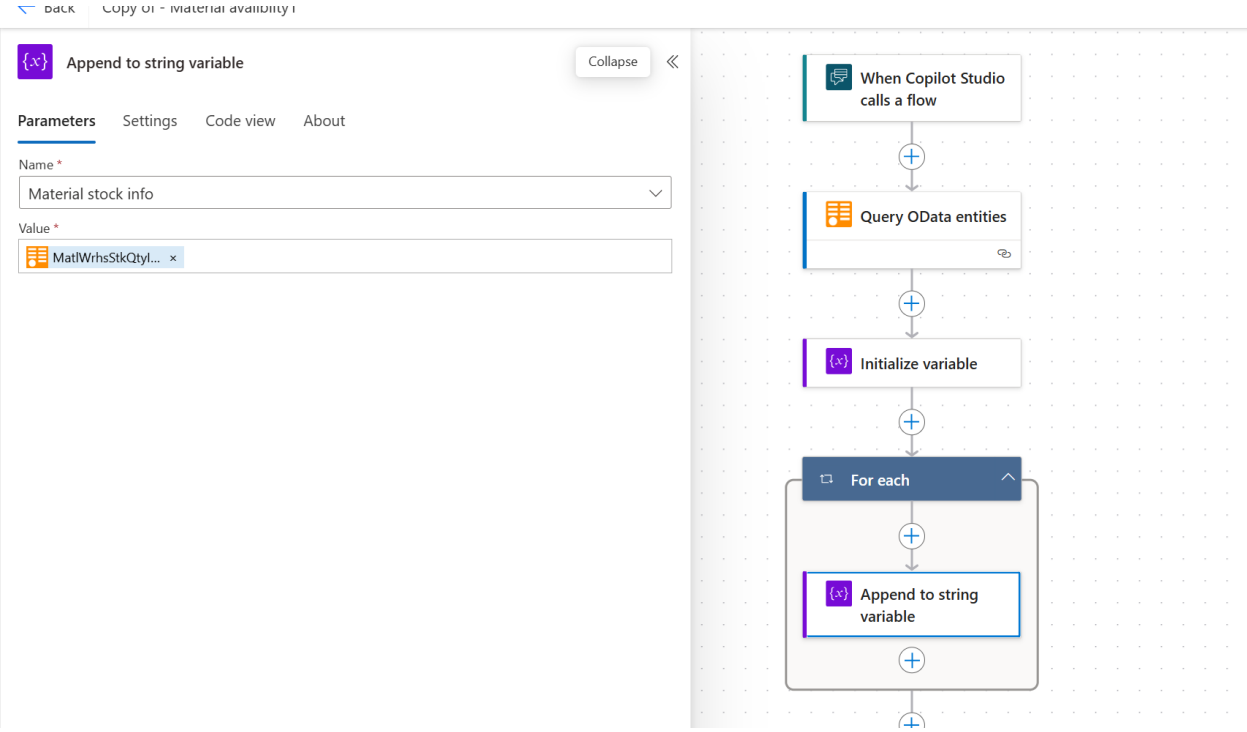
The next step is to initialize a variable to store the material stock information.



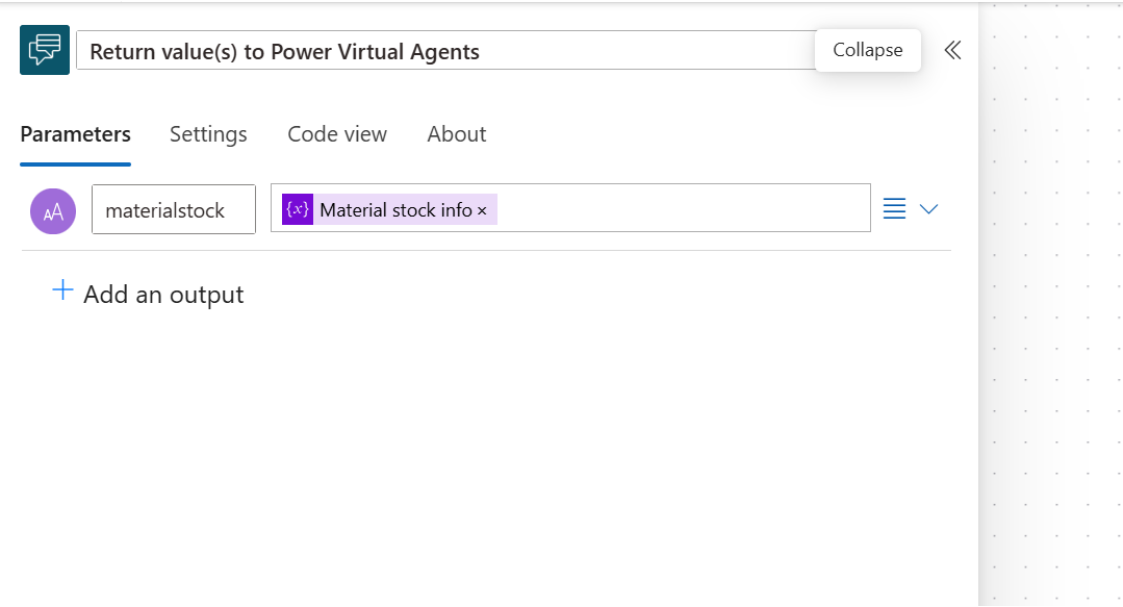
Next, loop over the response of the body and then append the material stock information to the variable created above.

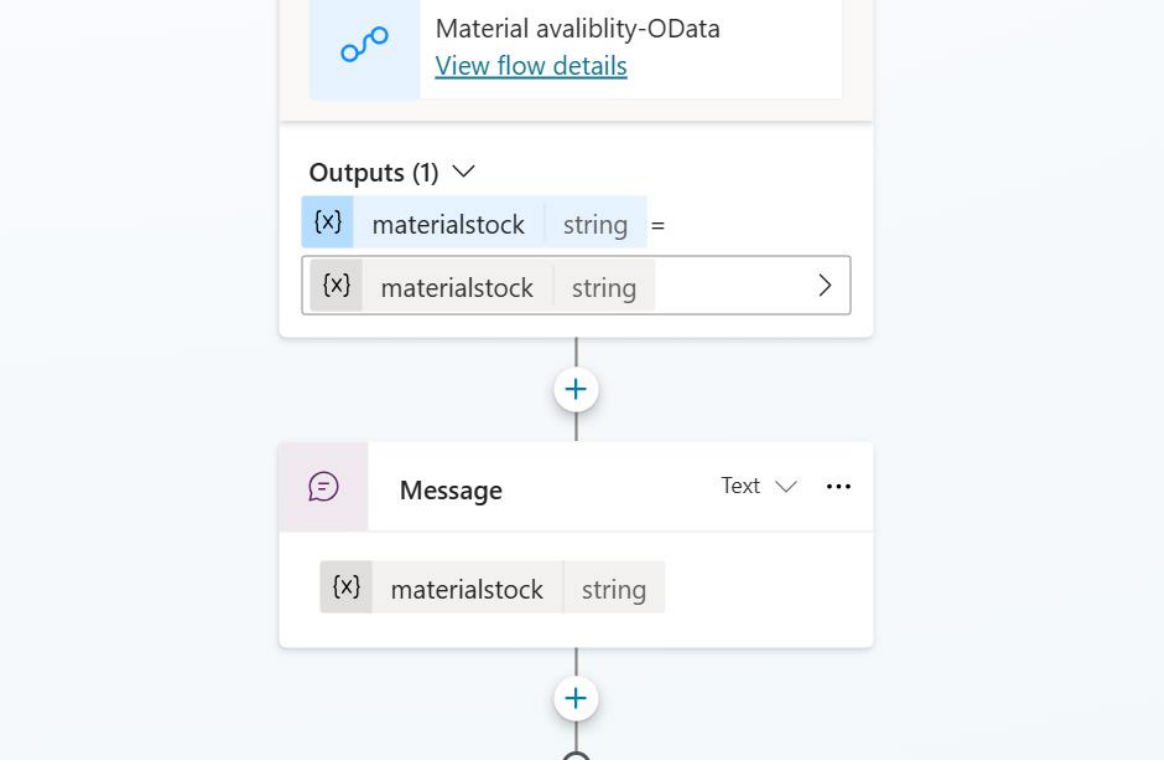


The variable to append to get the material stock information is the one shown below.



Once the material stock information has been retrieved, the value can be returned to Copilot Studio.

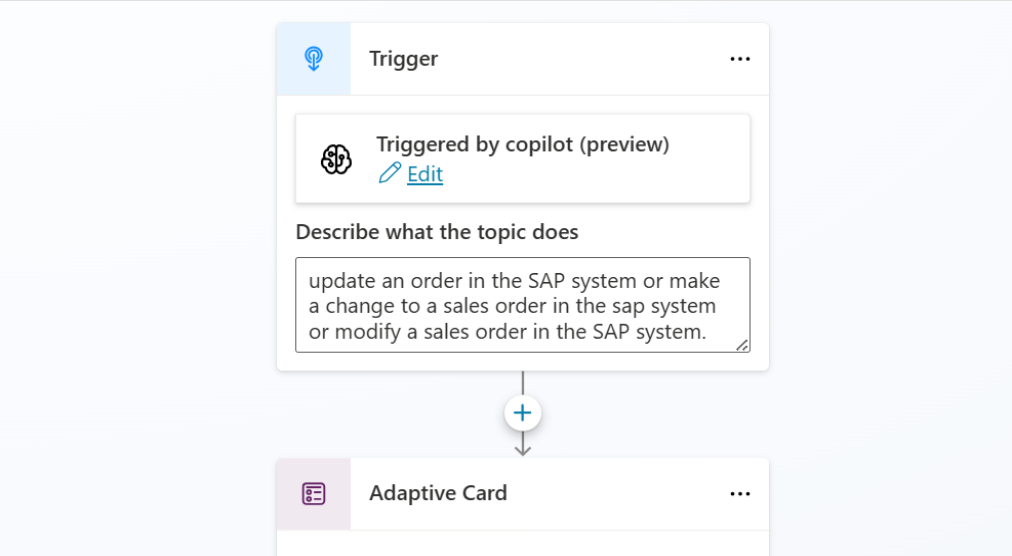




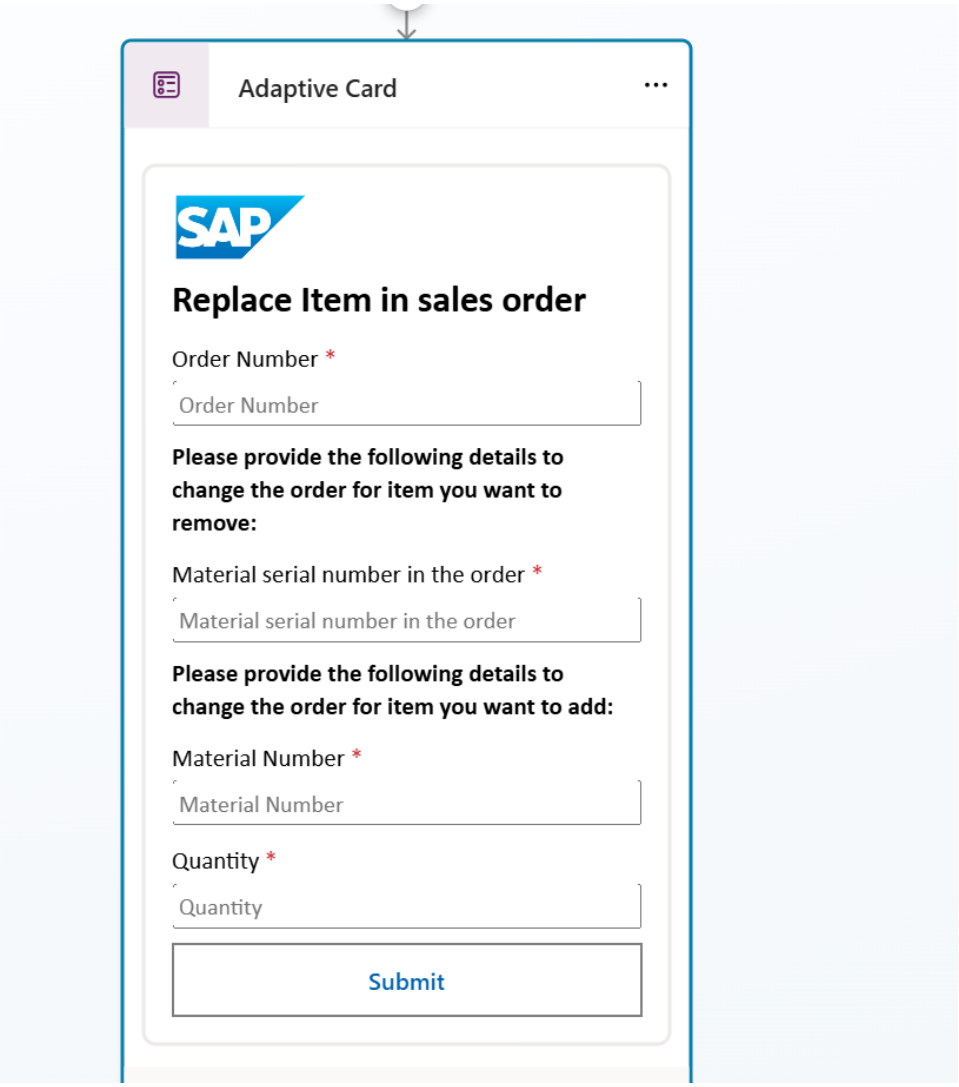
Once the user knows that the material is in stock, the user wants to update the sales order with the new material:

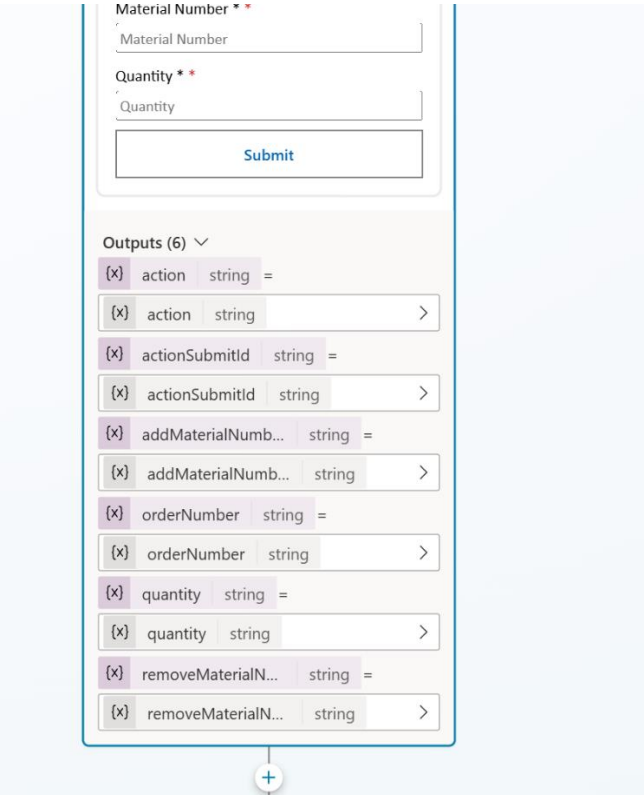
For this we will create a new Copilot Studio topic called “Modify sales order”.

Here is the description for it:

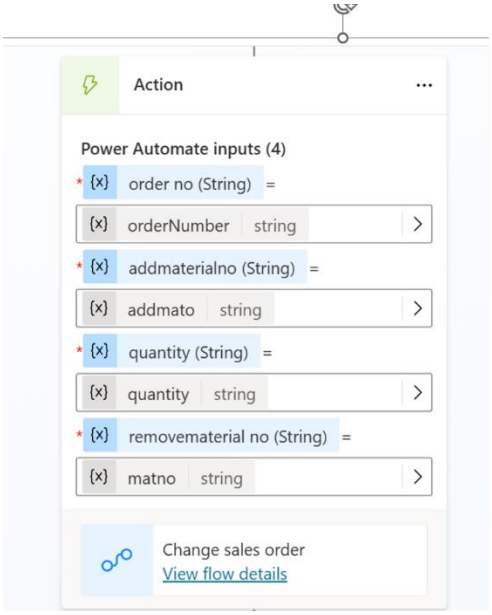


Next, we will create an adaptive card that asks the user to enter all the required information to change the order as shown below, the code for this adaptive card is on GitHub under the name (SalesorderchangeAdaptivecard.json").

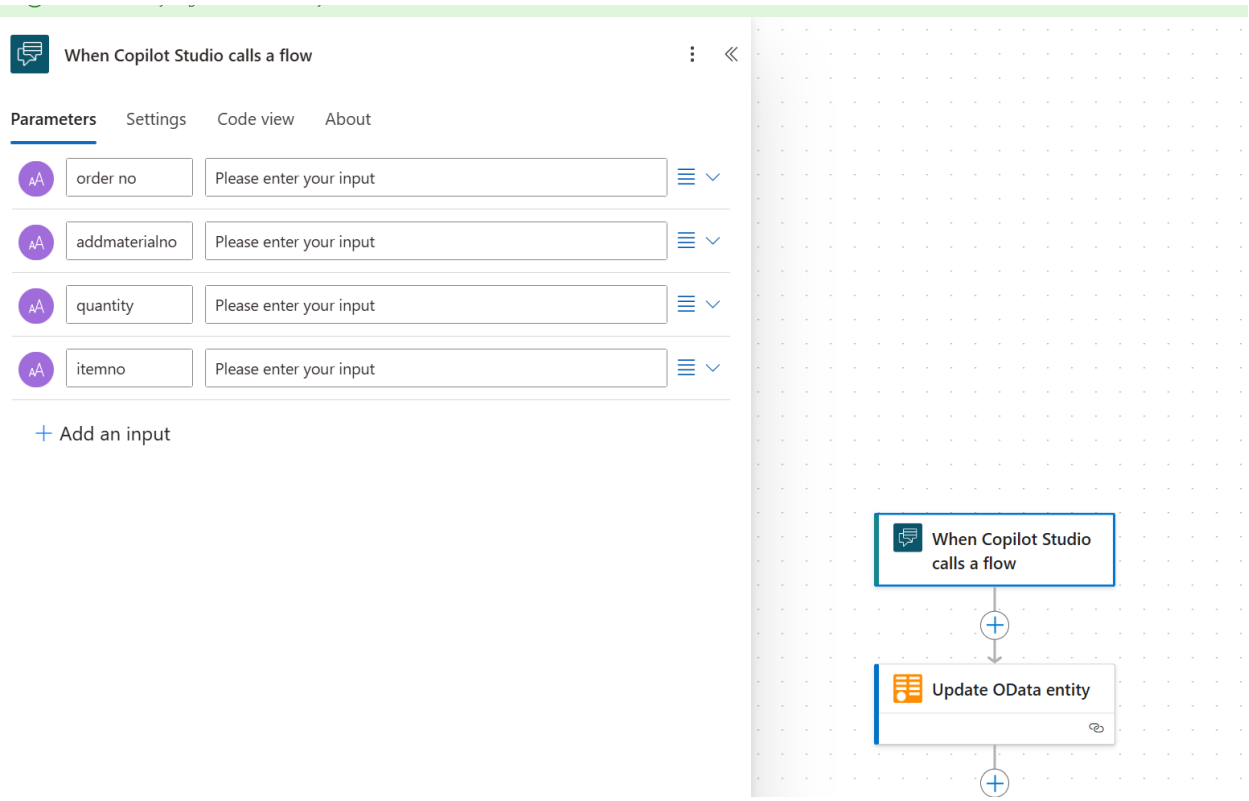

Adaptive Card
Replace Item in sales order
Order Number *
Order Number
Please provide the following details to change the order for item you want to remove:
Material serial number in the order *
Material serial number in the order
Please provide the following details to change the order for item you want to add:
Material Number *
Material Number
Quantity *
Quantity
Submit



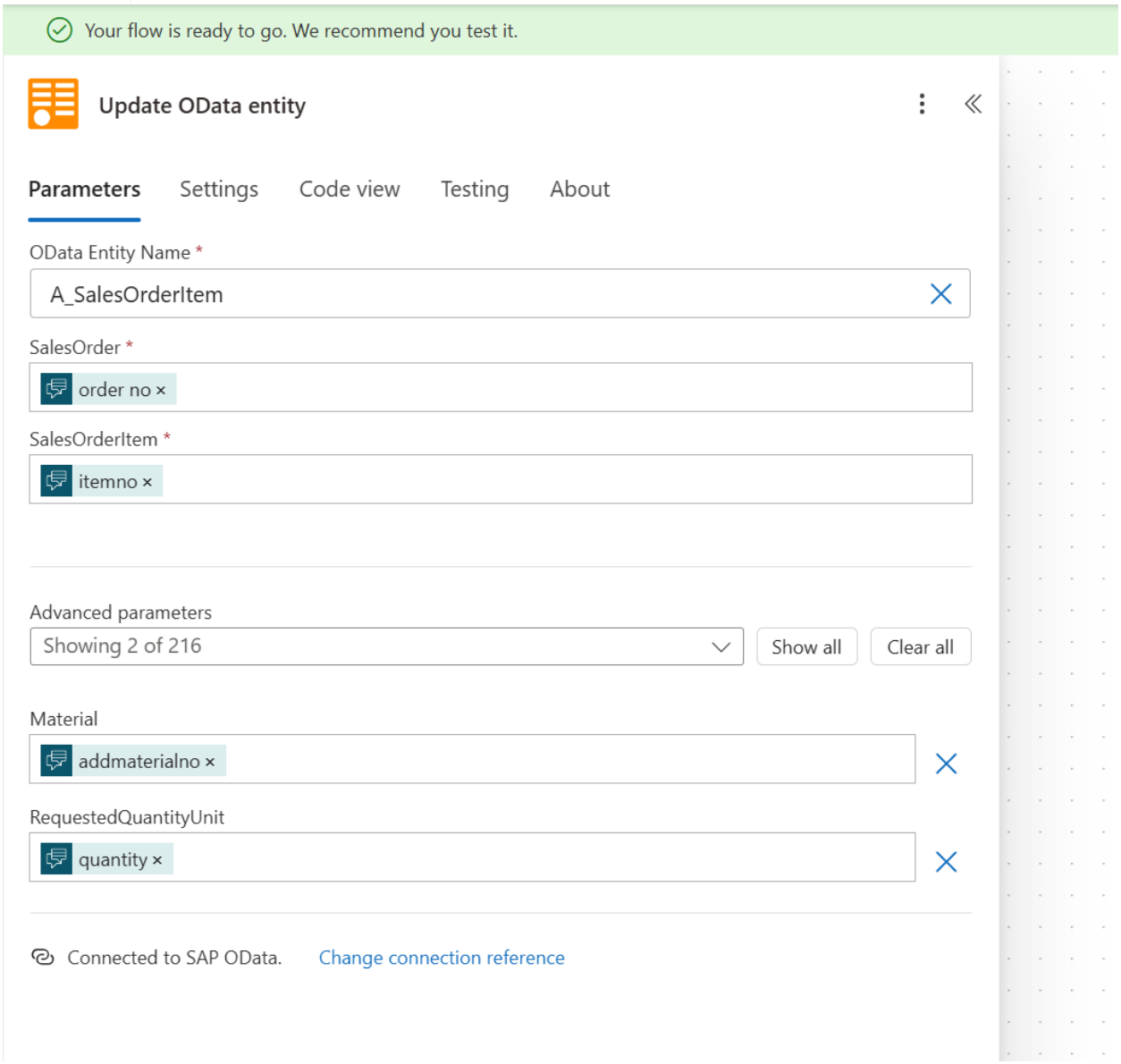
Now we will call a Power Automate flow ‘Change sales order -OData to modify the salesorder using the API_SalesOrder_SRV with the inputs shown:



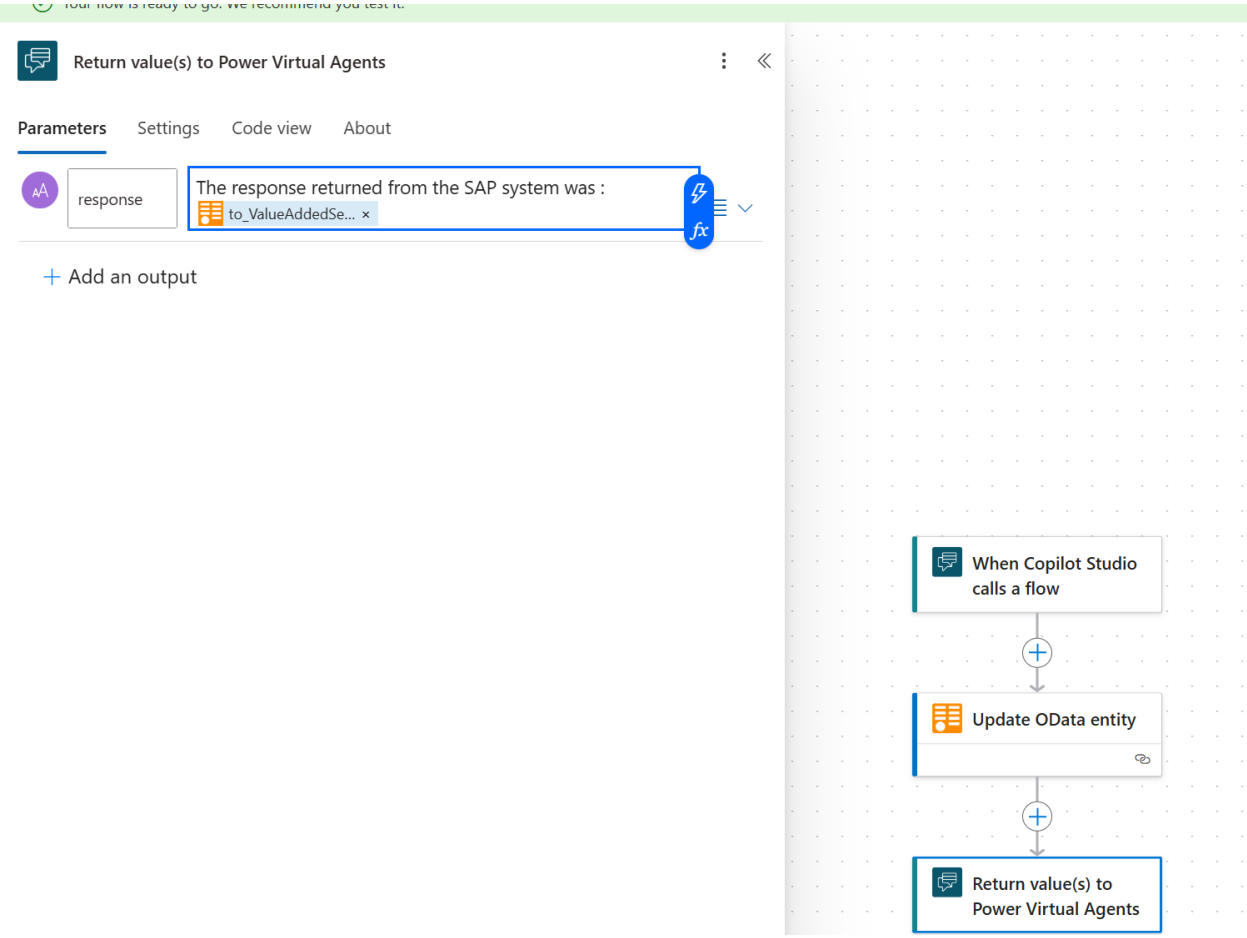
The Power Automate flow is called : Change sales order - OData and the inputs from the Copilot Studio are as shown below:



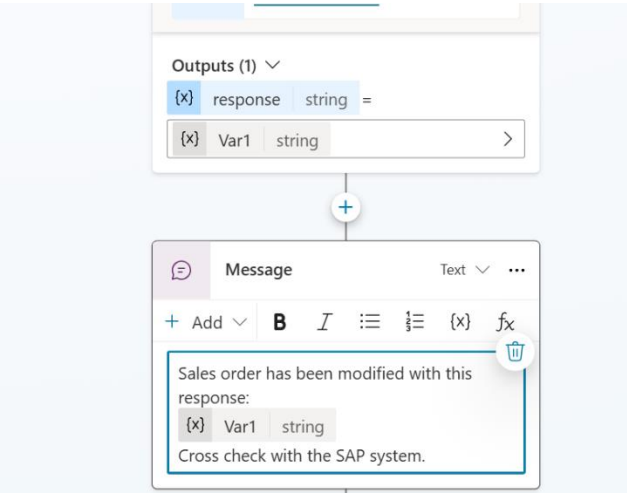
Next we select the SAP OData update action and call the API_Salesorder_SRV and select the entity A_SalesOrderItem, and place the variables as shown below:


Your flow is ready to go. We recommend you test it.
Update OData entity
Parameters
Settings
Code view
Testing
About
OData Entity Name *
A_SalesOrderItem
SalesOrder *
order no ×
SalesOrderItem *
itemno ×
Advanced parameters
Showing 2 of 216
Show all
Clear all
Material
addmaterialno ×
RequestedQuantityUnit
quantity ×
Connected to SAP OData.
Change connection reference

Finally we return the body of the response to the CopilotStudio as shown below.



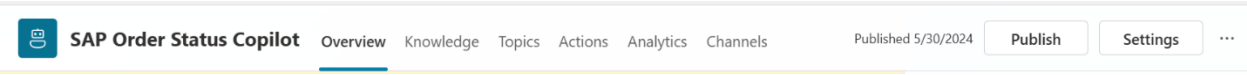
Finally, back in the Copilot Studio, we update the user and let them know whether their sales order has been updated or not.



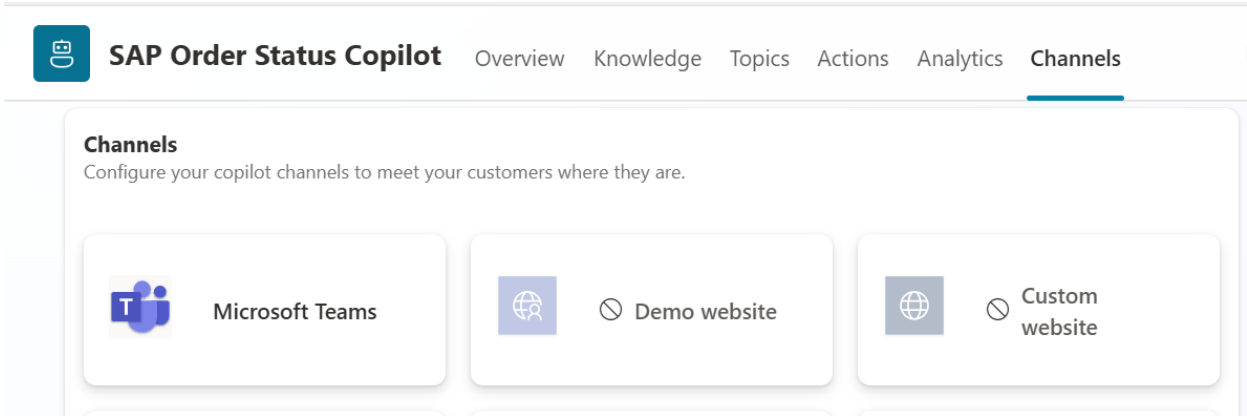
.

This concludes the flow.

You can now save and publish the Copilot by clicking “Publish”



Finally, launch the Copilot/bot on Teams by navigating to the Channels tab and clicking on Teams!



This scenario is just a starting point that will give you all the raw materials and components to build more scenarios in different domains.