# 

Logic Apps and Azure Functions

Hands On Lab

Adapted by Wagner Silveira

Original Material from Tomasso Groenendijk, Eldert Grootenboer, Steef-Jan Wiggers, Rob Fox created for the  
Global Integration Bootcamp 2017

Contents

[Naming Conventions 3](#_Toc478119428)

[Lab 4 - Logic Apps + Azure Functions 4](#_Toc478119429)

[Objective 4](#_Toc478119430)

[Prerequisites 4](#_Toc478119431)

[Steps 5](#_Toc478119432)

[Create Storage Account 6](#_Toc478119433)

[Create Strorage Container 8](#_Toc478119434)

[Create Storage Table 10](#_Toc478119435)

[Provision the Function App 15](#_Toc478119436)

[Building a Function 17](#_Toc478119437)

[Provision a Logic App 22](#_Toc478119438)

[Building a Logic App Definition 25](#_Toc478119439)

[Test the Solution 40](#_Toc478119440)

# Naming Conventions

As each resource in Azure should have a unique name, we advise to use the following naming convention, for easier identification and to avoid conflicts, as some resources, like storage accounts and Azure Functions require a unique public name.

|  |  |  |
| --- | --- | --- |
| Resource | Pattern | Example |
| Resource Grou | gab2017akl-rg-<ini> | Gab2017akl-rg-ws |
| Logic App | gab2017akl-logic-<ini>-processorder | gab2017akl-logic-ws-processorder |
| Storage Account | gab2017aklstor<ini> | gab2017aklstorws |
| Azure Function | gab2017akl-func-<ini> | gab2017akl-func-ws |

# Logic Apps and Azure Lab

## Scenario

In this hands-on lab, customers will be sending orders to be processed. **Logic App** will handle the logic, checking the total amount of the invoice. In cases where the customer placed a large order (over **$50000.00**), this will be redirected to a sales representative to contact the customer to verify the order.

If the order order is correct, the invoice will be emailed to the customer. The **Logic App** will then call a **function**, to check if the customer is elegible for a discount – based on a discount table stored in a **storage** **table**. The discount information will be stored in a **blob** **storage**, which will be used later to refund the customer.



### Prerequisites

* Azure Subscription
* Azure Storage Explorer: <http://storageexplorer.com/>
* Google Chrome Postman
* Outlook.com account
* TableCustomerDiscount.csv which can be downloaded from [here](https://www.globalintegrationbootcamp.com/wp-content/uploads/2017/03/TableCustomerDiscount.csv)

### Steps

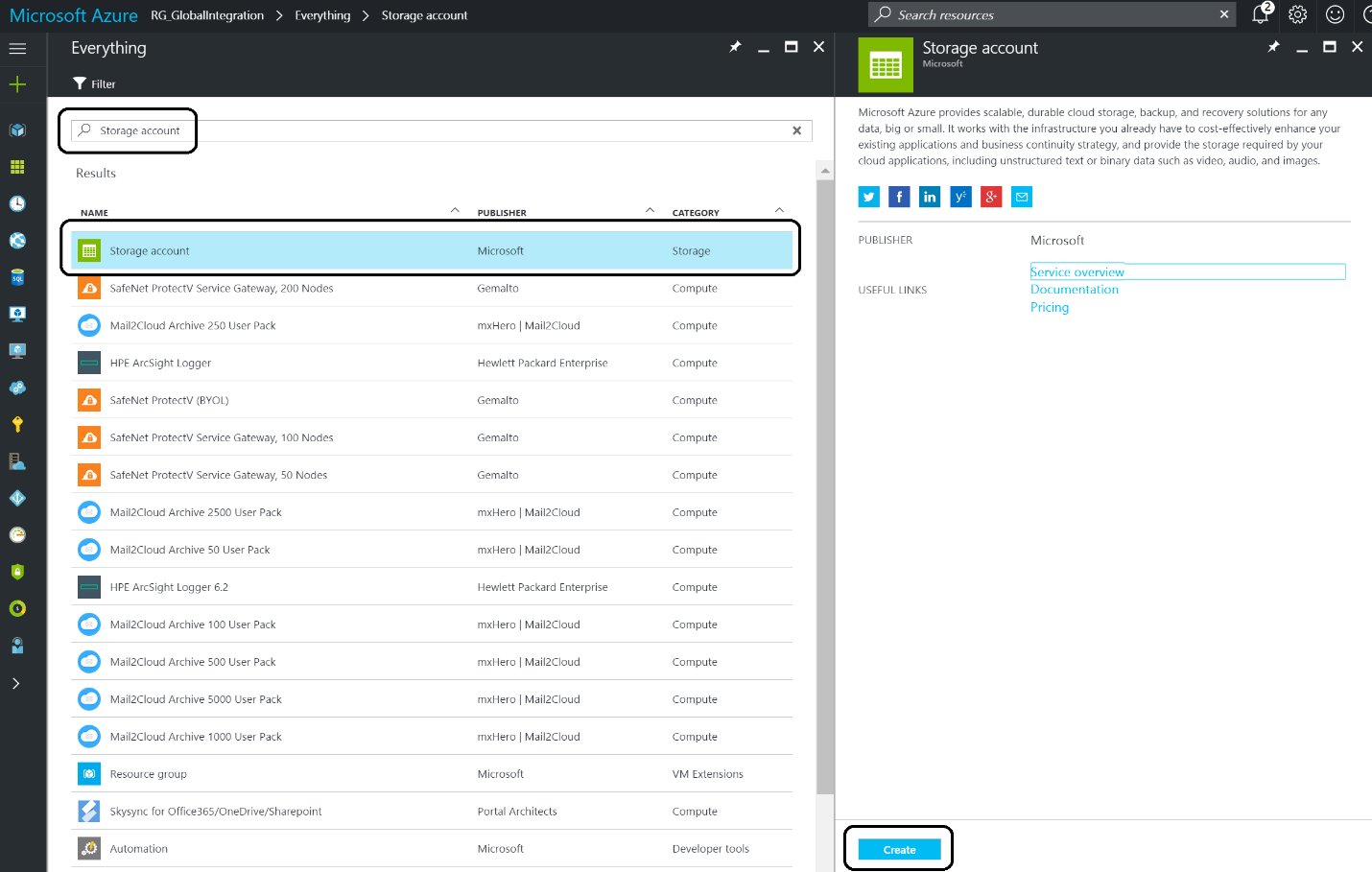
To build the solution in this lab you have to follow the steps described in this section. From a high level view the steps are:

1. Provision a storage account
2. Create Storage Blob Container
3. Create Storage Table
4. Provision the Function App
5. Build a custom function
6. Provision a Logic App
7. Build Logic App Definition
8. Test the Solution

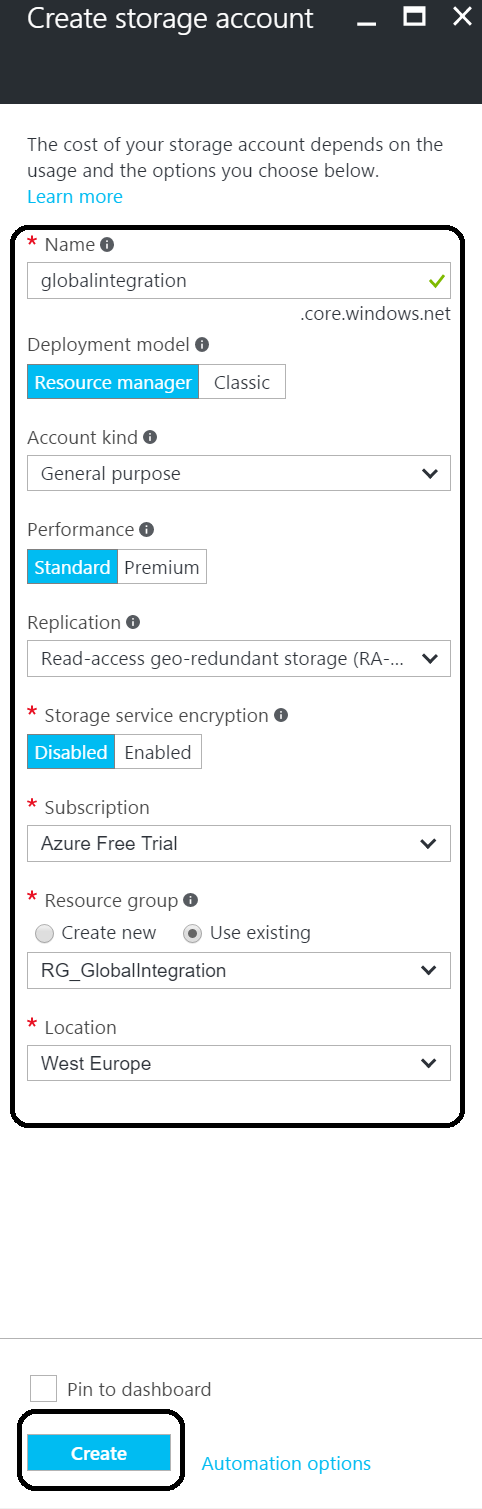
## Provison the Storage Account

The first step in building the solution in this lab is to provision a storage account in Azure. We will be needing storage for setting up our reference table (Table Storage) and storing the order request message in Blob Storage.

1. Go to the Azure Portal: <https://portal.azure.com/>
2. Login into the Azure portal with your account.
3. In the Market Place enter storage account and select it from the list as shown below.



1. Click **Create**.

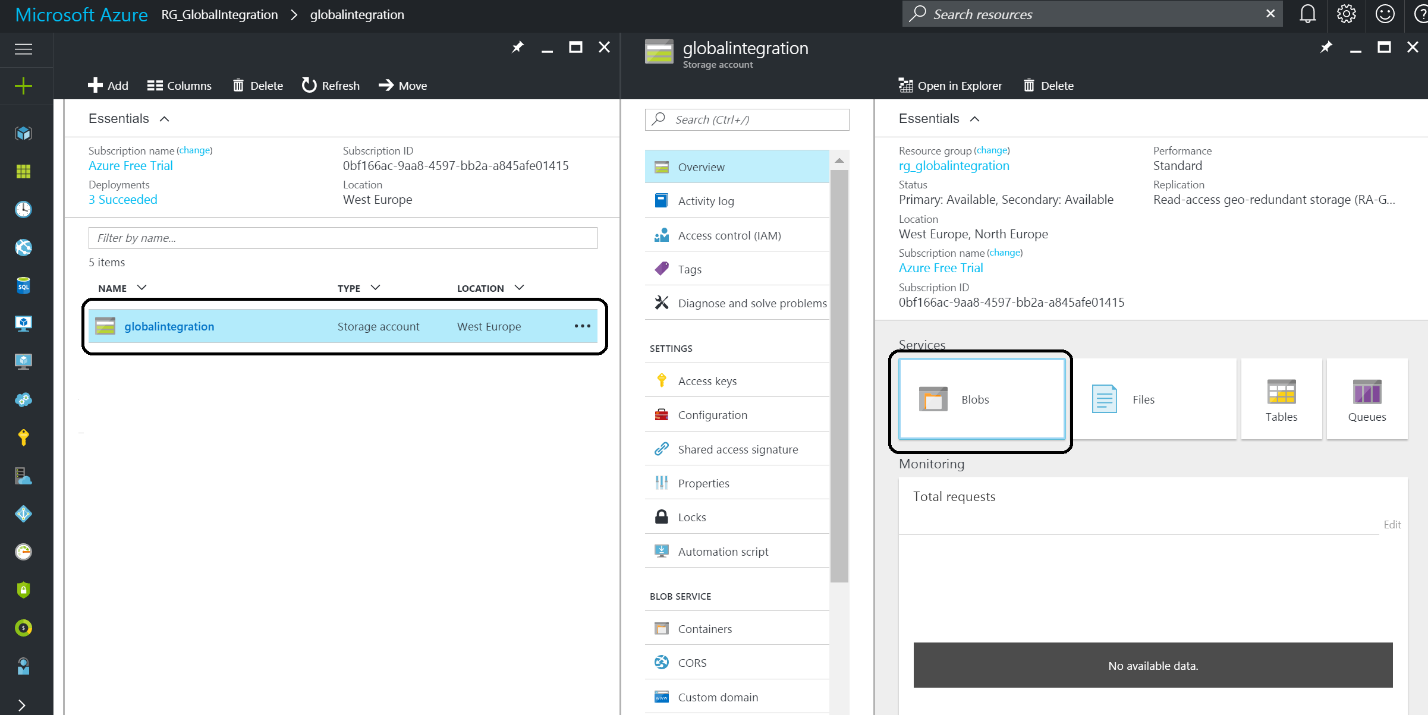


1. Specify the name- **gab2017aklstor<ini>**, the **Resource Group – gab2017akl-rg-<ini>** (you can create a new one here if you haven’t created a resource group yet) and a **location**. Subsequently, click on **Create**.

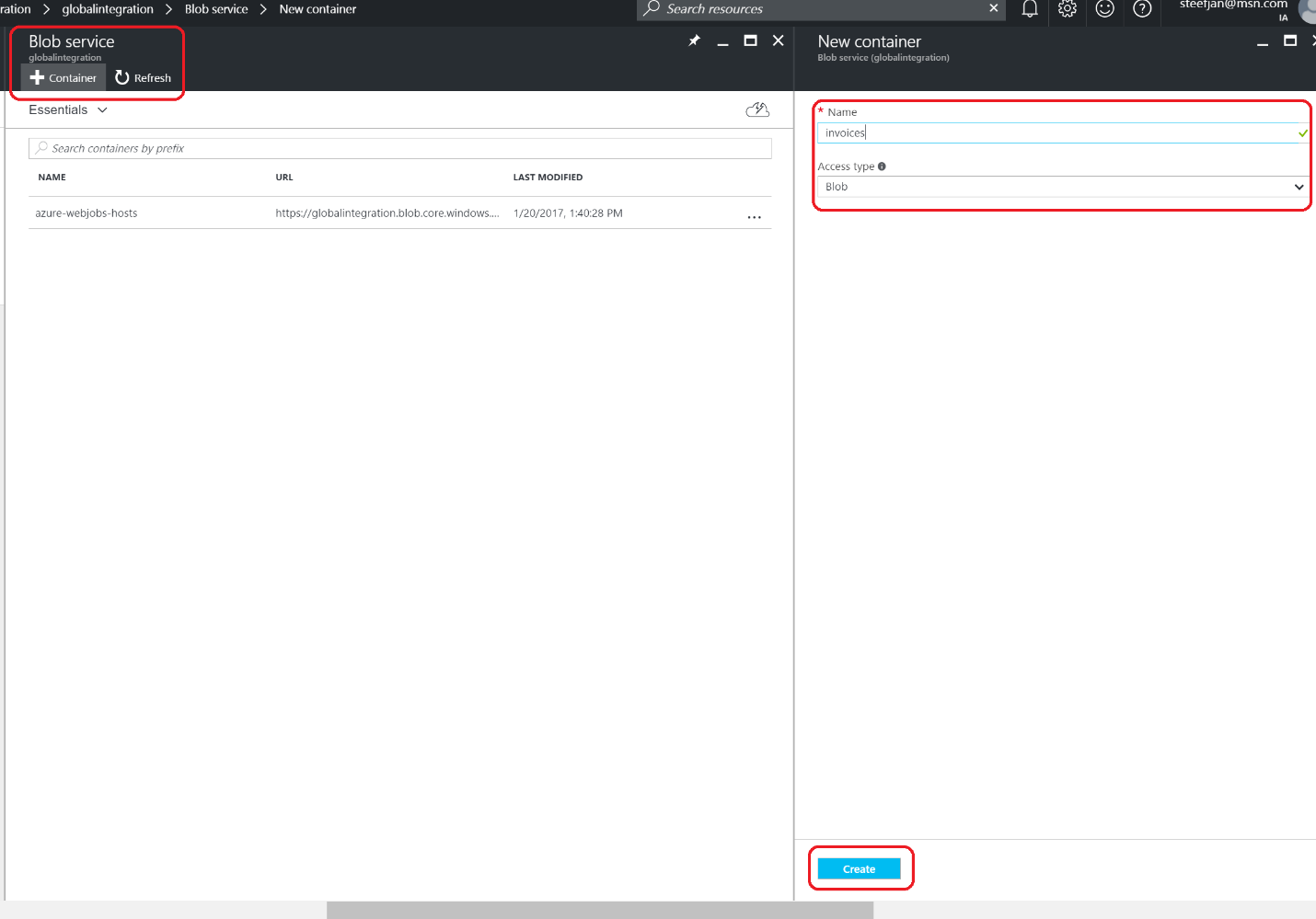
## Create Strorage Container

Once the storage account has been provisioned you can navigate to it and click on it.

1. In the storage account click on **Blobs**.



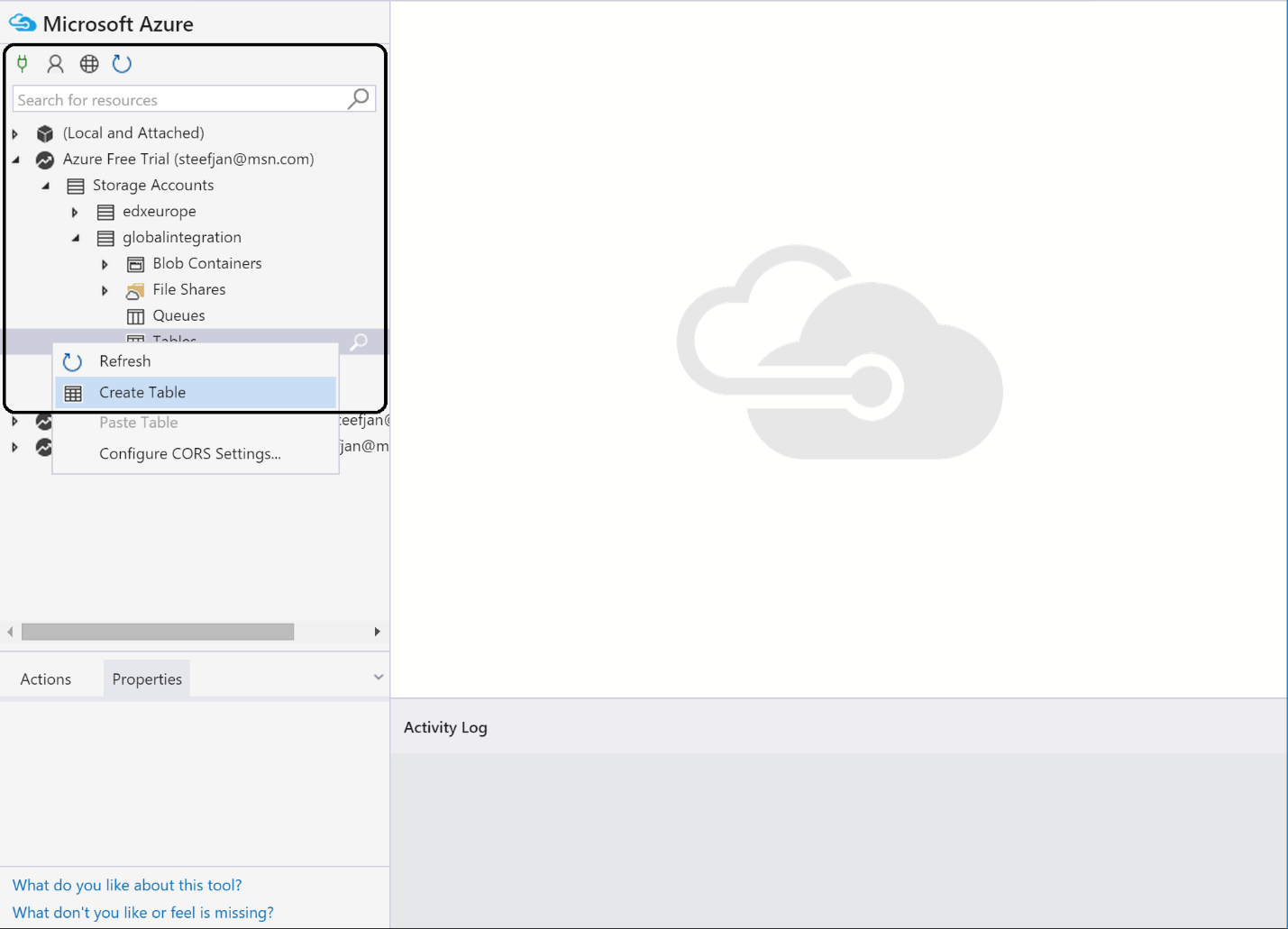
1. Click on **+ Container** and specify the name – **invoices -** and Access Type- **Blob**.



## Create Storage Table

To create a storage table will use the **Azure Storage Explorer**, which can be downloaded from <http://storageexplorer.com/>.

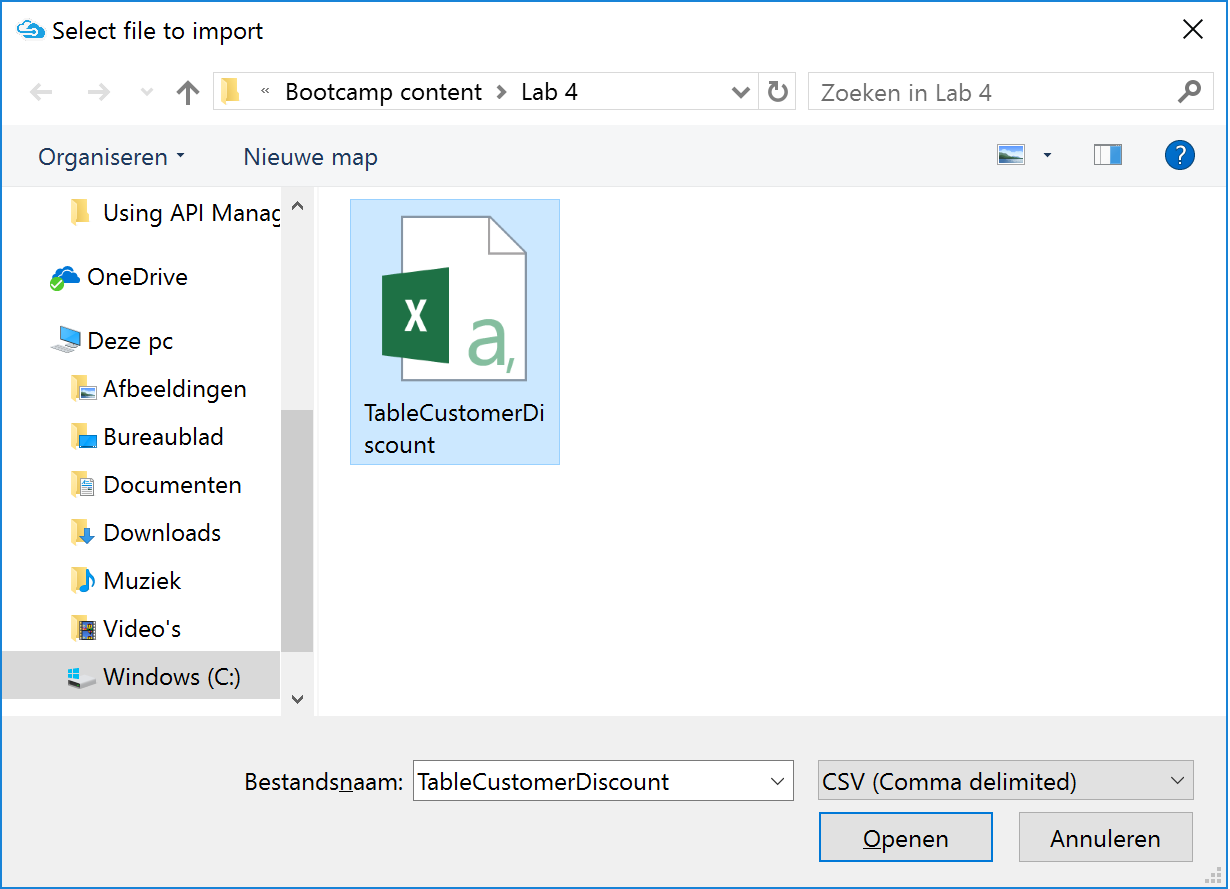
1. Install the tool, and login into your subscription.
2. Navigate to your storage account.
3. Select Tables
4. Right click Tables and click Create Table.



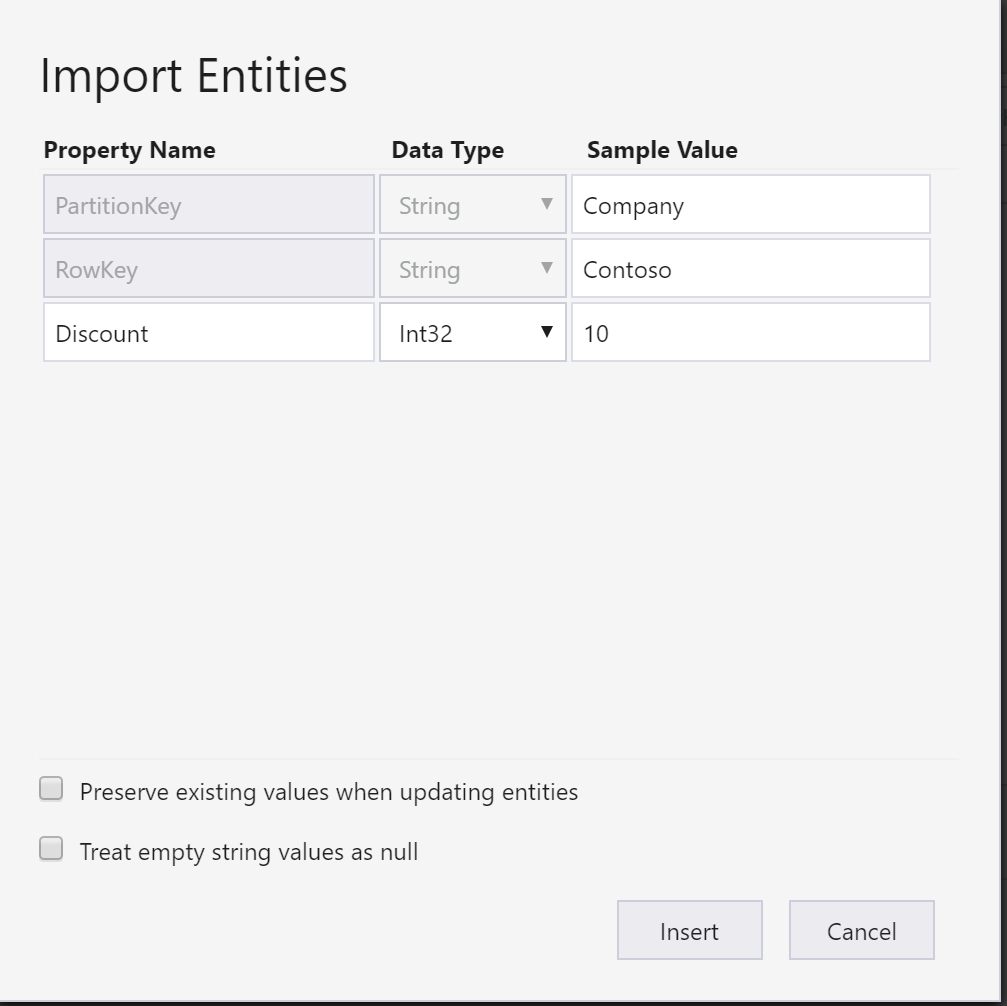
1. Specify a name for the table - **CustomerDiscount**.
2. Select the **table**.
3. Click on **Import** **Entities** from file.



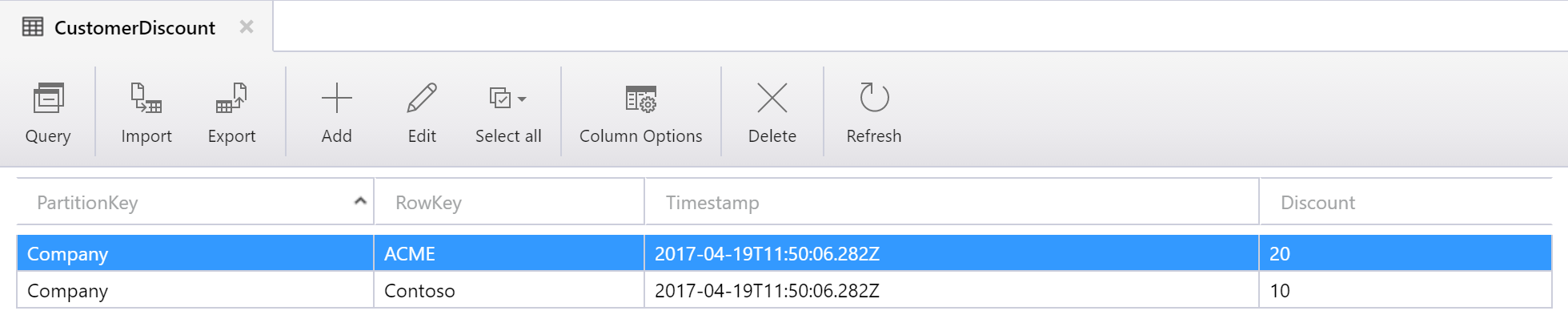
1. Select the **TableCustomerDiscount.csv** you downloaded previously.



1. When prompted with the window below, click insert:



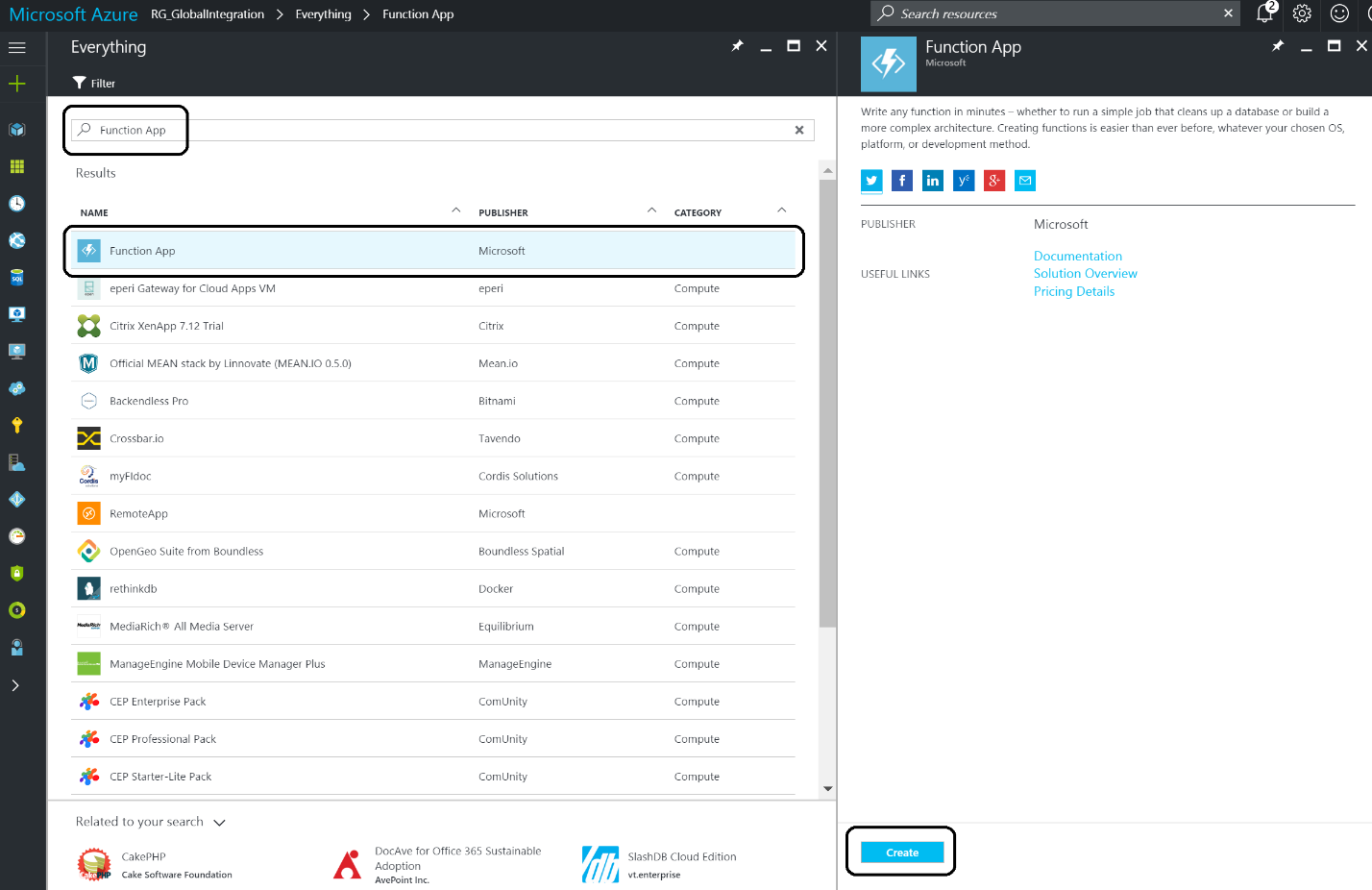
1. **Verify the Table** was loaded with data.

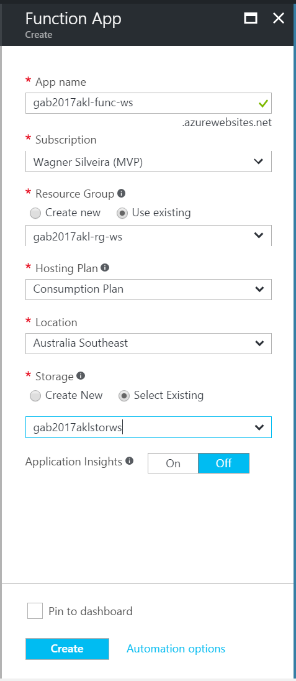


## Provision the Function App

The third step in building up our solution is provision a **Function App**. A **Function App** is a container for your functions. Those functions can be built with a browser using either C#, JavaScript, or some of the other languages. The code you create can be run and tested in the function app environment.

1. In the Market Place enter **Function** **App**.
2. Select the **Function** **App** and **Click Create**.



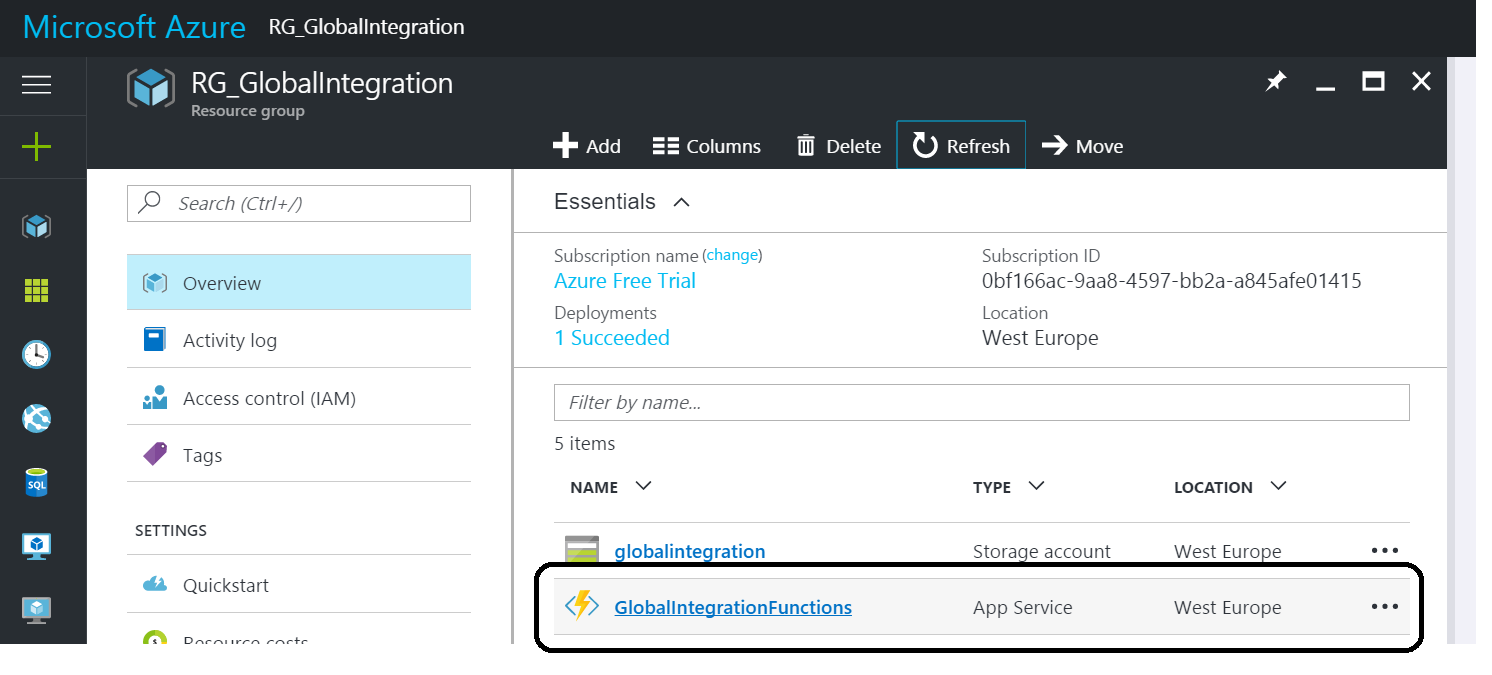


1. Specify the following details:
   1. Name: **gab2017akl-func-<ini>**
   2. Resource Group – select existing: **gab2017akl-rg-<ini>**
   3. Hosting Plan: **Consumption plan**
   4. Location: <will be auto populated by the Resource Group>
   5. Storage – select existing: **gab2017aklstor<ini>**

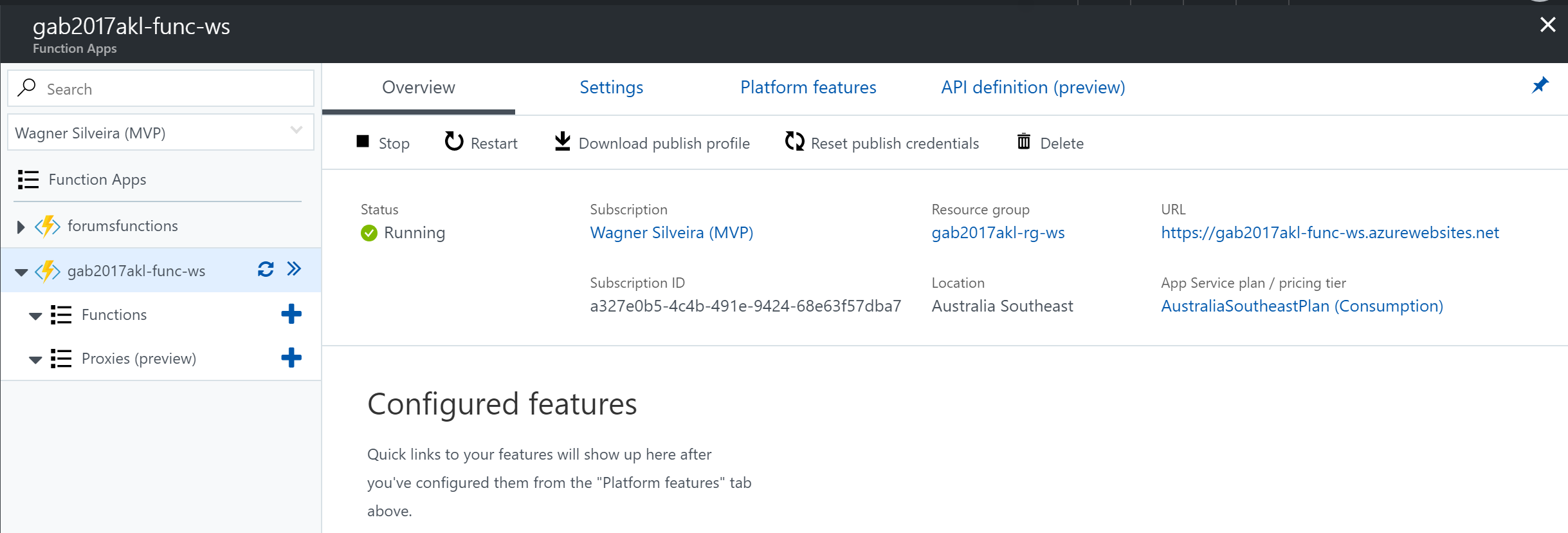
## Building a Function

Once the function app is provisioned you can add function to it, build and test it.

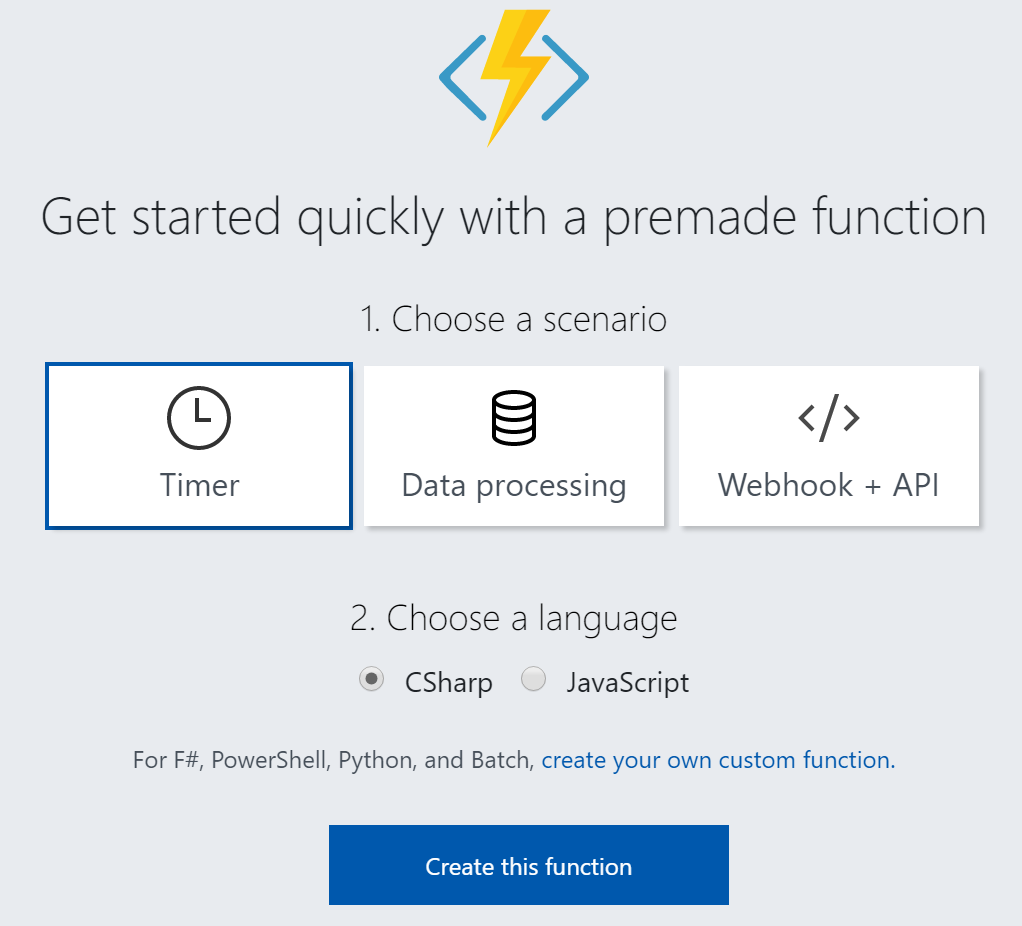
1. Navigate to the **Function App** from the Resource Group:



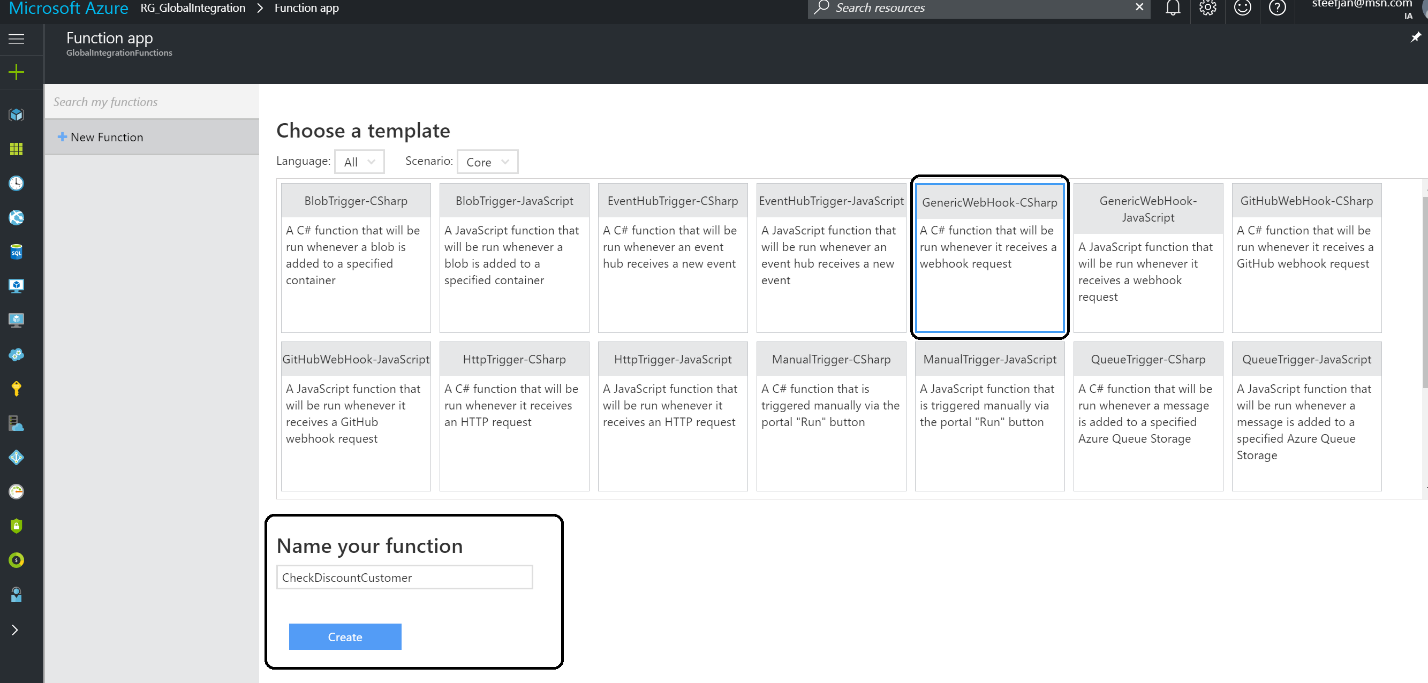
1. In the function app click **Functions +**.



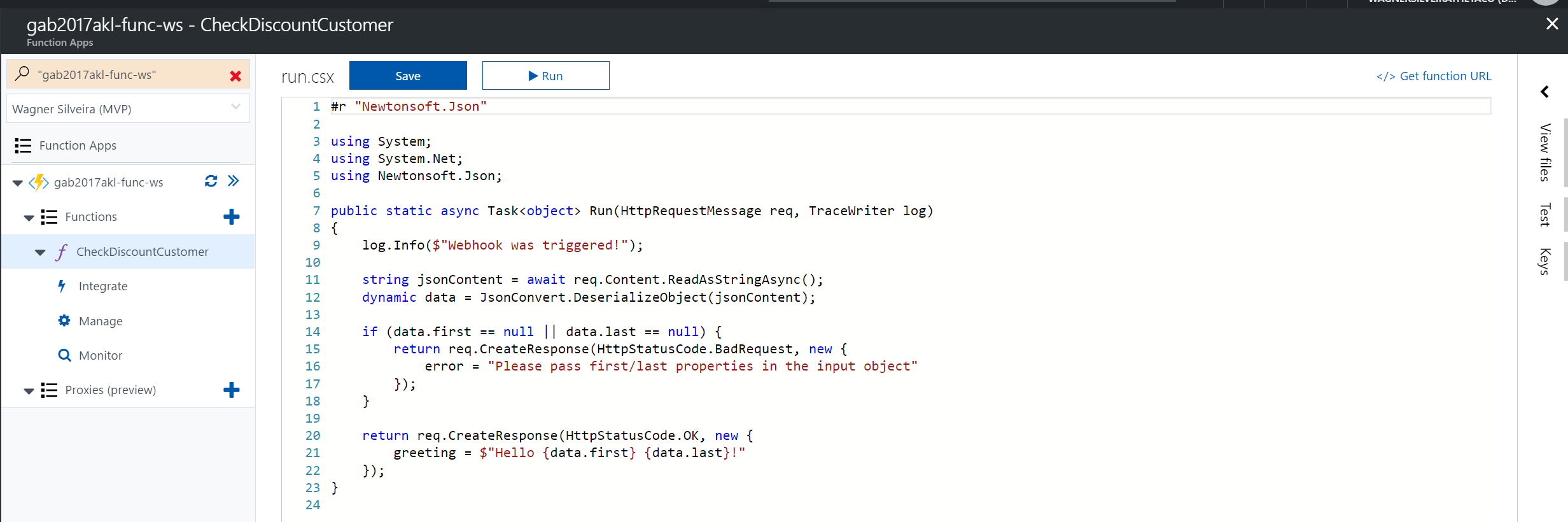
1. In the quick start function, click on **create your own custom function**:



1. Select **GenericWebHook-CSharp** and specify **CheckDiscountCustomer** as the a name for your function.



1. Now a new pane will appear with some default code.



1. Replace the code with the code below:

#

r "Microsoft.WindowsAzure.Storage"#

r "Newtonsoft.Json"

**using** System**;**

**using** System**.**Net**;**

**using** Microsoft**.**WindowsAzure**.**Storage**;**

**using** Microsoft**.**WindowsAzure**.**Storage**.**Table**;**

**using** Newtonsoft**.**Json**;**

**public** static **async** Task **<** **object** **>** Run**(**HttpRequestMessage req**,** TraceWriter log**)** **{**

log**.**Info**(**$ "Webhook was triggered!"**);**

int **?** discount **=** 0**;**

string accountName **=** "**gab2017aklstor<ini>**"**;**

string accountKey **=** "**<storagekey>**"**;**

string jsonContent **=** **await** req**.**Content**.**ReadAsStringAsync**();**

**dynamic** data **=** JsonConvert**.**DeserializeObject**(**jsonContent**);**

string company **=** data**.**Company**;**

// Here we will connect to our storage account

**try** **{**

string storageAccountConnectionString **=** String**.**Format**(**"DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}"**,** accountName**,** accountKey**);**

CloudStorageAccount storageAccount **=** CloudStorageAccount**.**Parse**(**storageAccountConnectionString**);**

CloudTableClient tableClient **=** storageAccount**.**CreateCloudTableClient**();**

CloudTable table **=** tableClient**.**GetTableReference**(**"CustomerDiscount"**);**

//Create a filter expression

var tableQuery **=** **new** TableQuery **<** DynamicTableEntity **>** **();**

tableQuery**.**FilterString **=** TableQuery**.**CombineFilters**(**

TableQuery**.**GenerateFilterCondition**(**"PartitionKey"**,** QueryComparisons**.**Equal**,** "Company"**),**

TableOperators**.**And**,**

TableQuery**.**GenerateFilterCondition**(**"RowKey"**,** QueryComparisons**.**Equal**,** company**));**

// Loop through the results, displaying information about the entity.

**foreach(**DynamicTableEntity entity **in** table**.**ExecuteQuery**(**tableQuery**))** **{**

var item **=** entity**.**Properties**;**

discount **=** item**[**"Discount"**].**Int32Value**;**

**}**

**}** **catch** **(**Exception ex**)** **{**

log**.**Info**(**$ "Webhook exception :" **+** ex**.**Message**);**

**}**

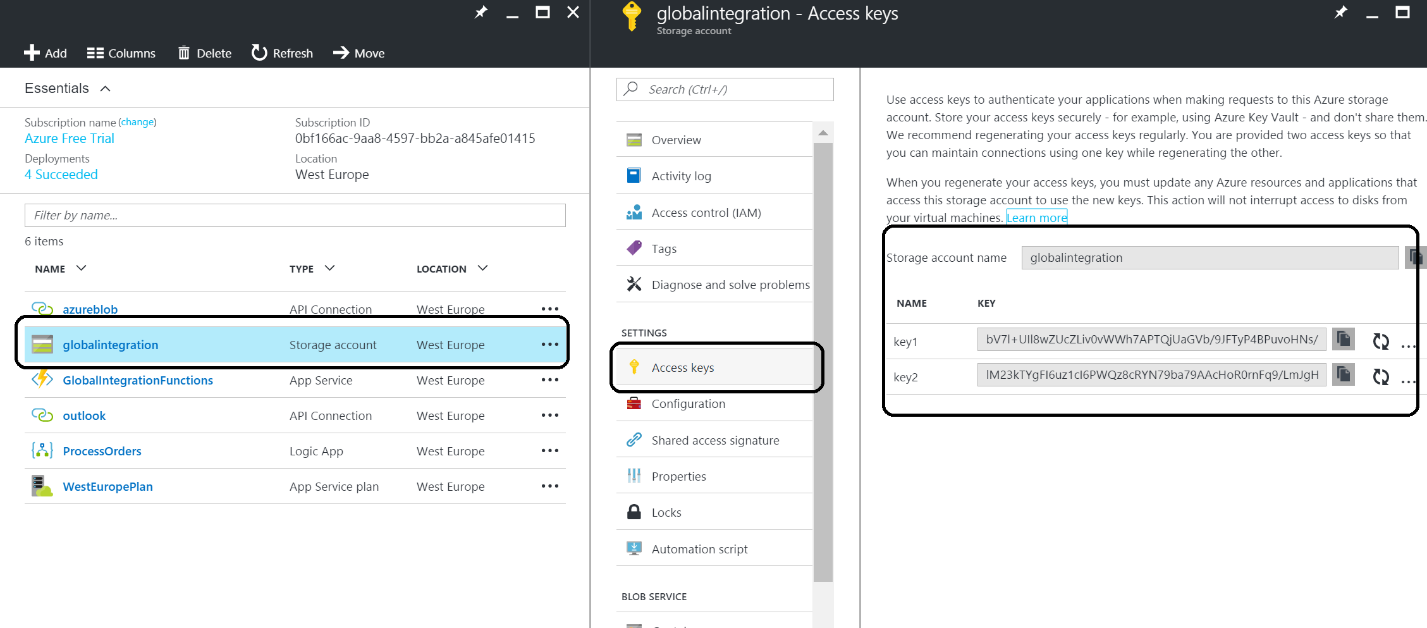
**return** req**.**CreateResponse**(**HttpStatusCode**.**OK**,** **new** **{**

discountValue **=** discount

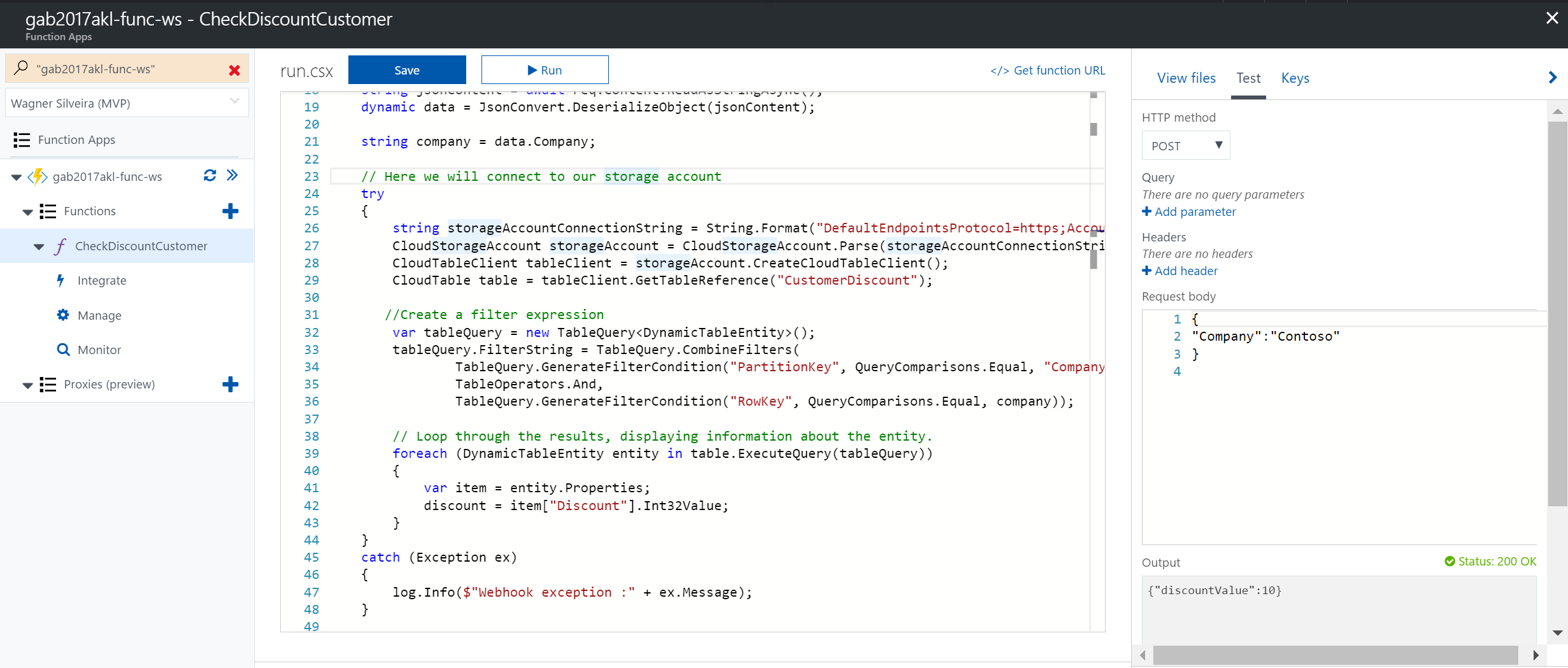
**});**

**}**

1. In the code above, replace **gab2017aklstor<ini>** with your storage account name, and **storagekey** the with the access key from your Storage account. You can find the key under Access Keys within your Storage Account blade.



1. Hit **Save** in the top bar.
2. Click on **Tests** in right top corner.

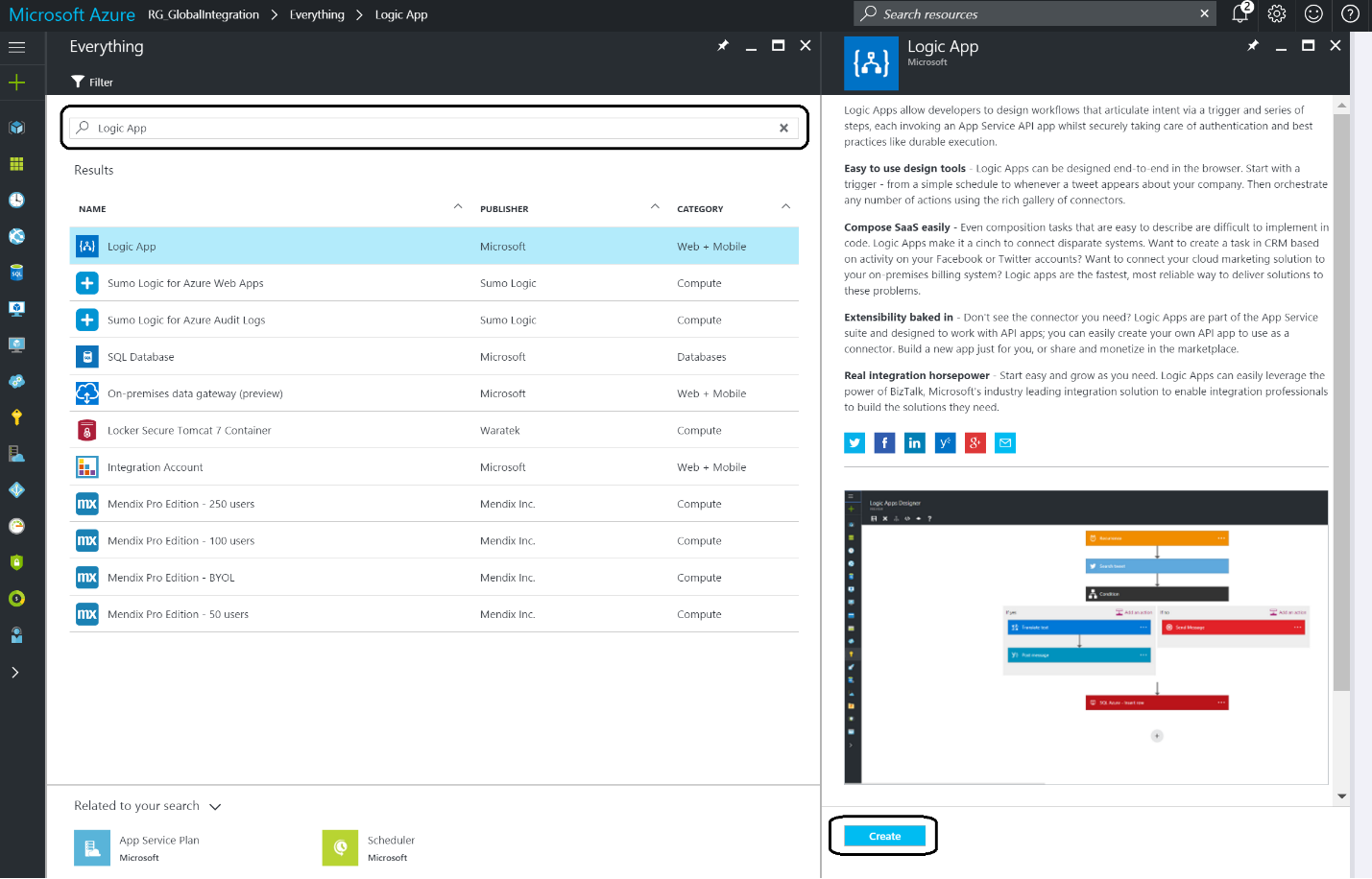


1. Change **Request body** to what is in the picture above.
2. Click **Run**.
3. Explore **Logs** and the **Output**.

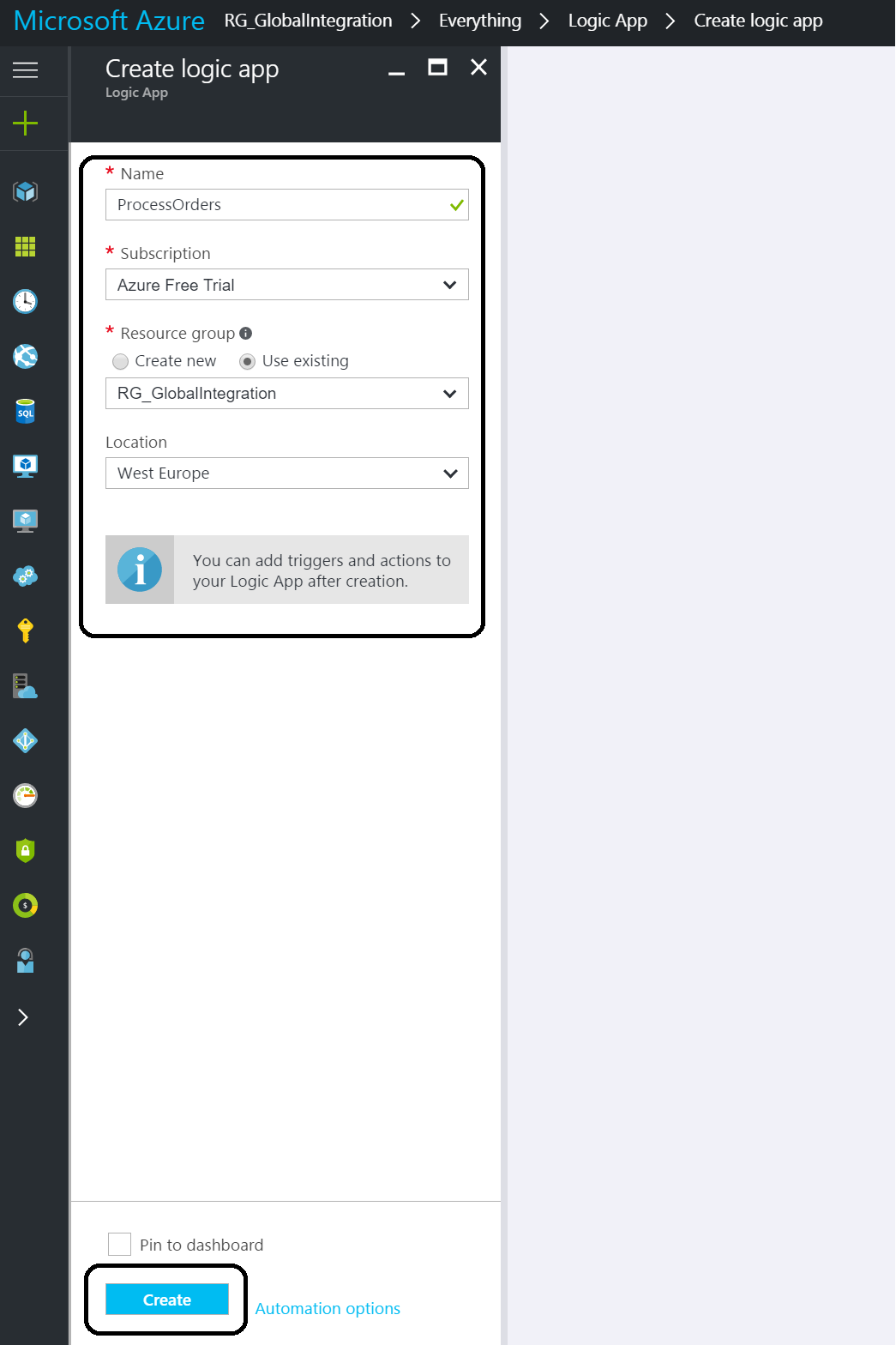
## Provision a Logic App

The following steps describe how to provision a Logic App.

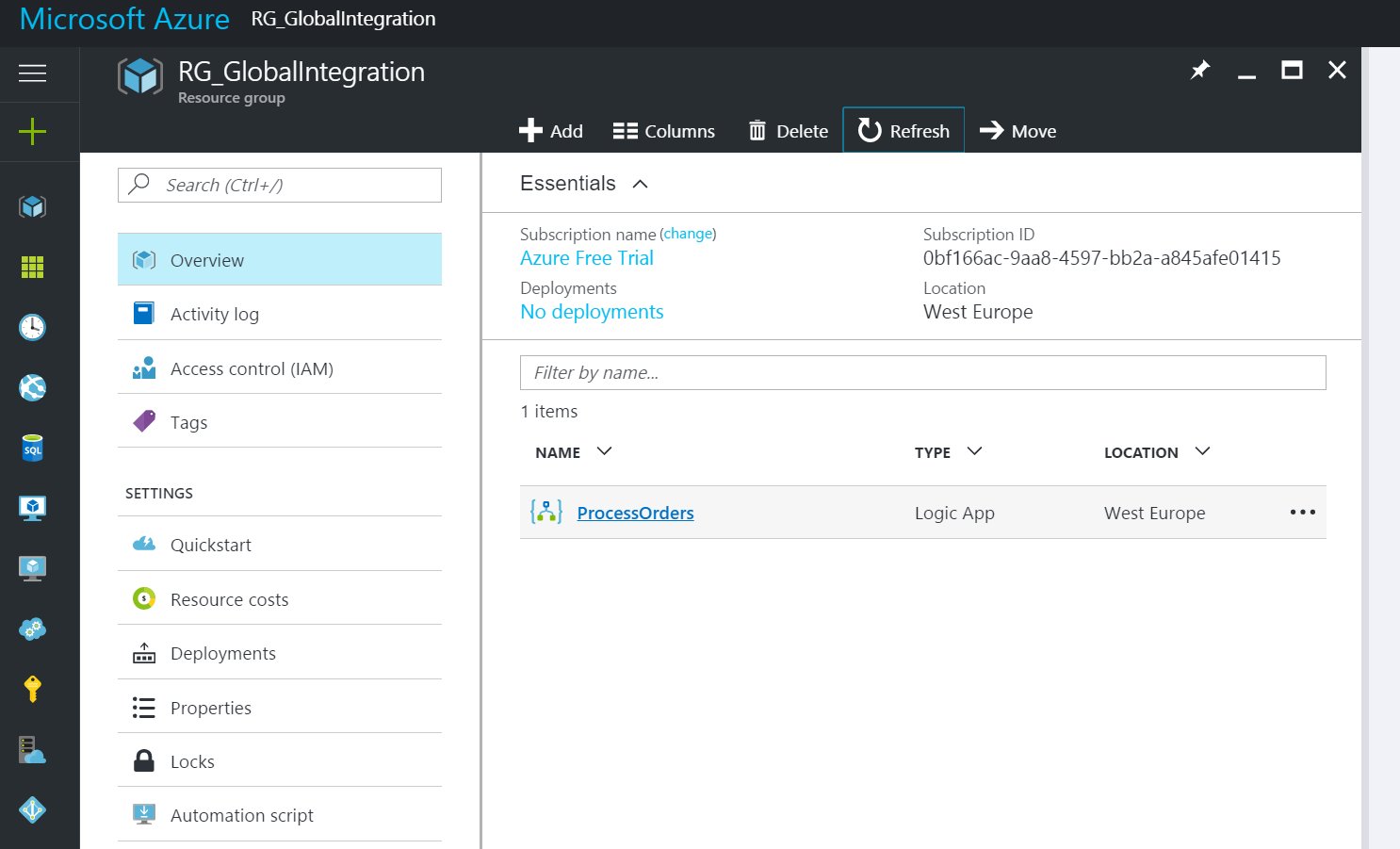
1. In the new Azure Portal click the **+**, navigate to **Web + Mobile** and subsequently click **Logic App**. Or in the search in the marketplace type Logic App. It’s easy to find the Azure Service in the portal to provision.



1. The next step is to specify name of your **Logic App**, the subscription (in case you would have multiple subscriptions), the resource group the Logic App should belong to and location i.e. which Azure datacenter. And subsequently decide if you want to see the Logic App on the dashboard or not and click **Create**. You wait until the Logic App is provisioned to be able to proceed with building up the flow (trigger and actions).



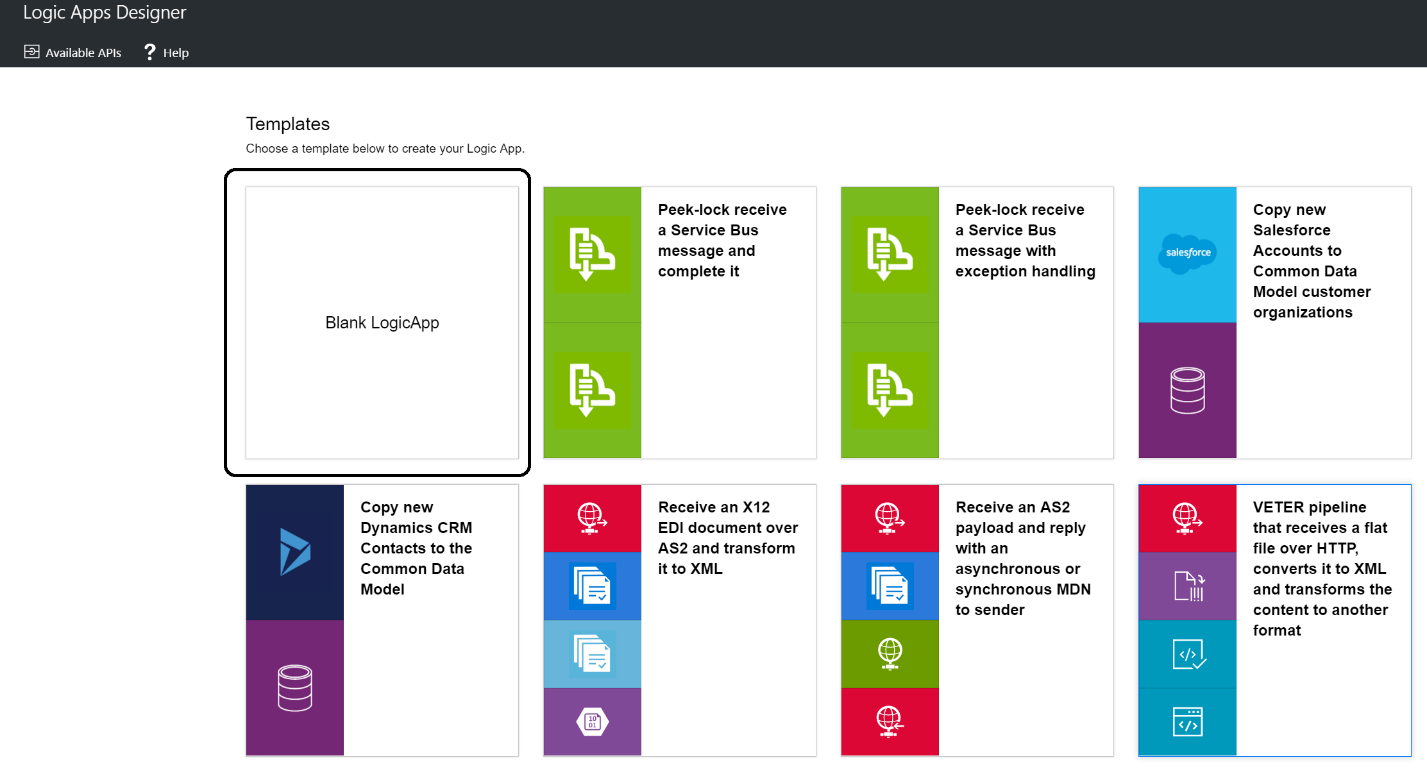
1. Once the **Logic App** is provisioned, you have setup a service, also known as iPaas (integration Platform as a Service). The **Logic App** is fully managed by Microsoft Azure and the only thing you need to do is add the logic i.e. specify the trigger and defining the actions, which we will do in the next step.



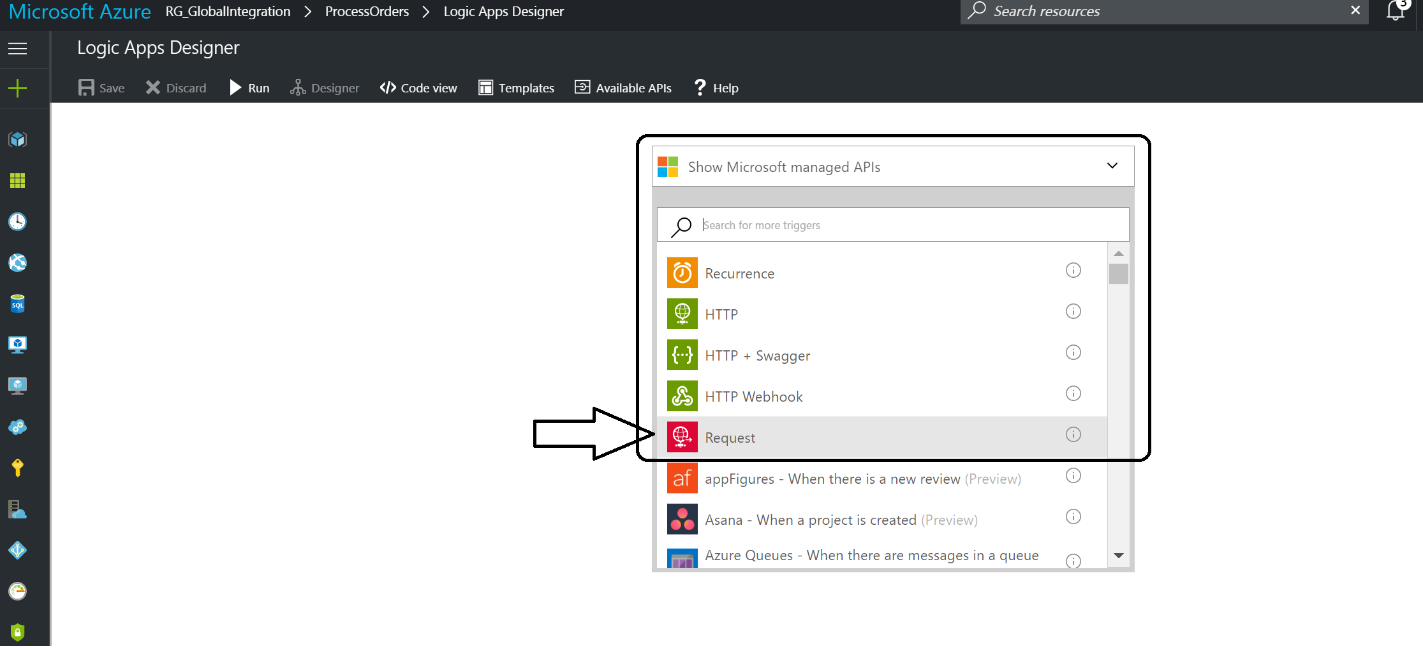
## Building a Logic App Definition

In the following steps, we will build our Logic App to support our solution.

1. Once the **Logic App** is provisioned, you can click on it, choose **Blank Logic App** and you will have access the **Logic Apps designer**.



1. In the Logic App designer, you can add a trigger. You select various triggers like **HTTP**, **recurrence**, **WebHook**, etcetera (see [Workflow Actions and Triggers](https://msdn.microsoft.com/library/azure/mt643939.aspx)). In this lab, it’s the **HTTP** **trigger** (Request) for testing purposes (or the **Service Bus** if you want to connect from the previous lab), which you have to provide with a schema of the payload it can accept/expect.



1. The schema can be generated directly from the designer, or using JsonSchema.net, a tool that automatically generates JSON schema from JSON according to the IETF JSON Schema Internet Draft Version 4. JSON Schema will be automatically generated in three formats: editable, code view, and string. Paste the code below in the JSON window as shown below:

**{**

"Order"**:{**

"Customer"**:{**

"Company"**:**"Motion10"**,**

"Email"**:**"eldert.grootenboer@motion10.com"**,**

"CustomerNumber"**:**"6f6d4907-23af-e611-80e5-5065f38a5a01"**,**

"Address"**:{**

"Street"**:**"Wilhelminakade 175"**,**

"City"**:**"Rotterdam"**,**

"PostalCode"**:**"3072AP"**,**

"Country"**:**"Netherlands"

**}**

**},**

"Products"**:{**

"Product"**:[**

**{**

"ProductNumber"**:**1000**,**

"Amount"**:**1**,**

"Price"**:**123.45

**},**

**{**

"ProductNumber"**:**2000**,**

"Amount"**:**5**,**

"Price"**:**456.78

**}**

**]**

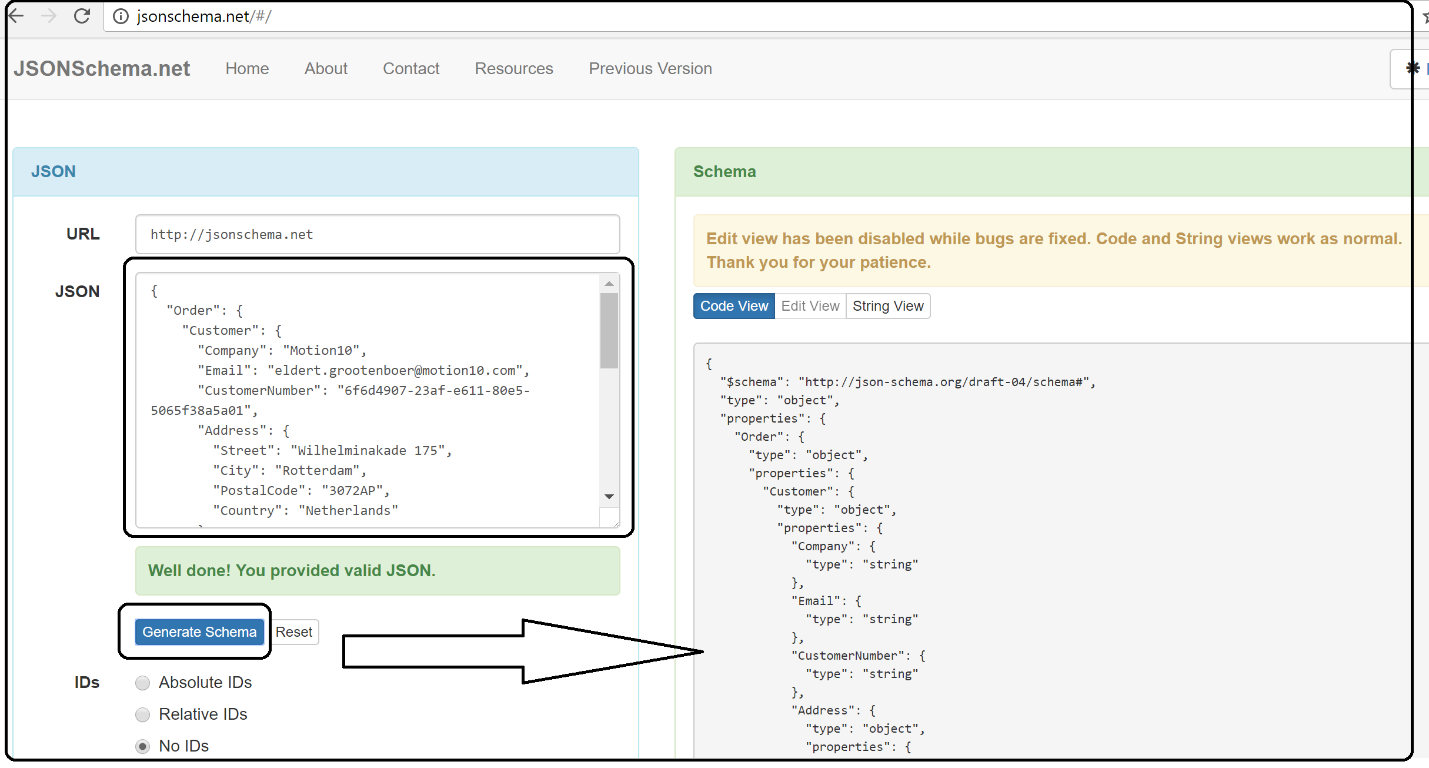
**},**

"OrderedDateTime"**:**"2016-11-20T14:26:00"**,**

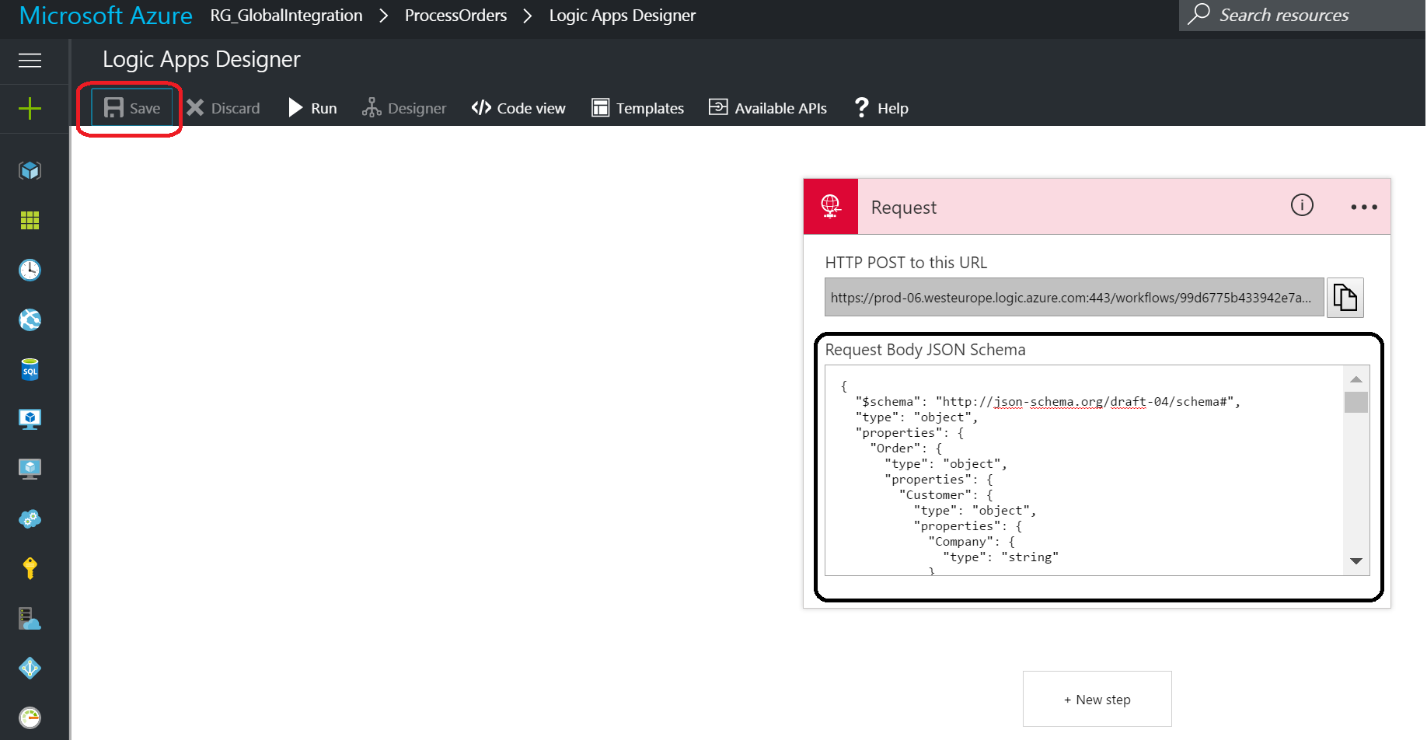
"TotalInvoiceAmount"**:**2407.35

**}**

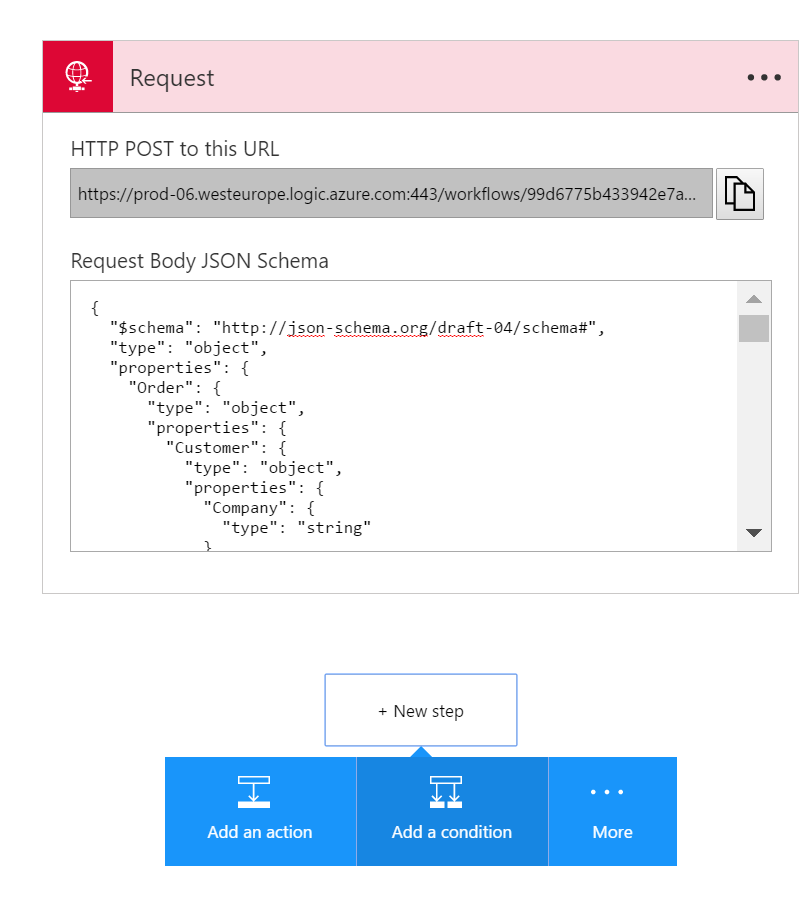
**}**



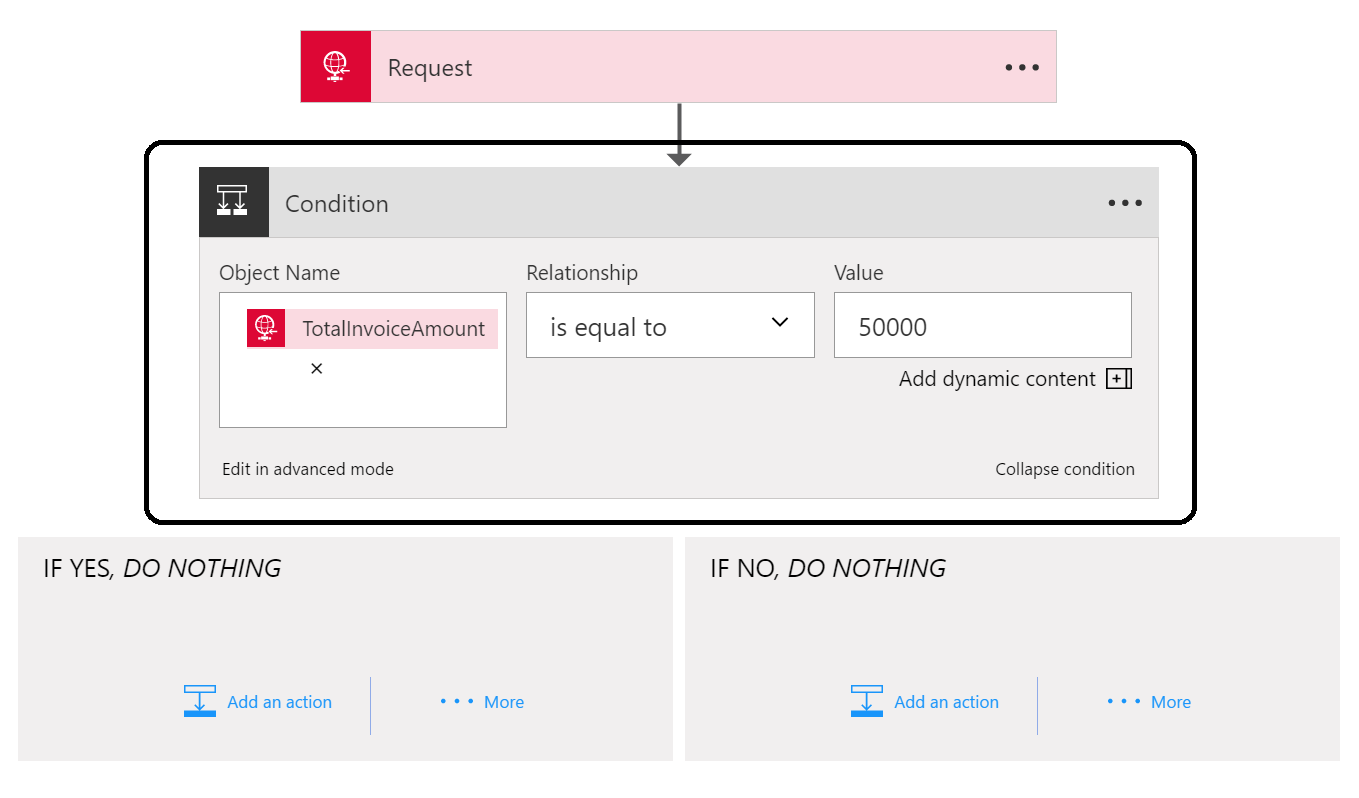
1. The generated schema can be pasted into the **Request Body JSON Schema** part of the HTTP Trigger. Note that the URL is generated after the Logic App has been saved for the first time.



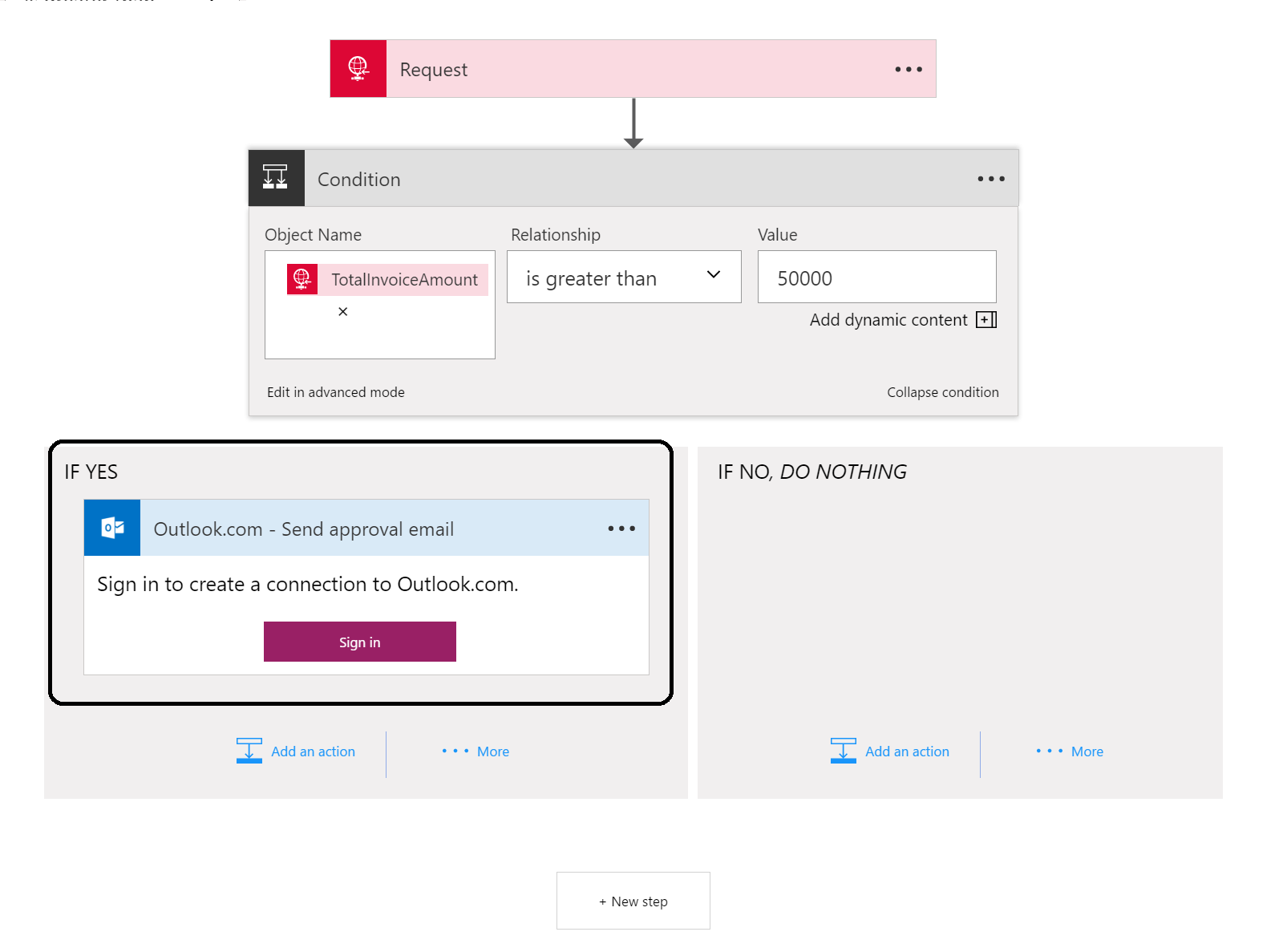
1. The next step is to add a **condition**.



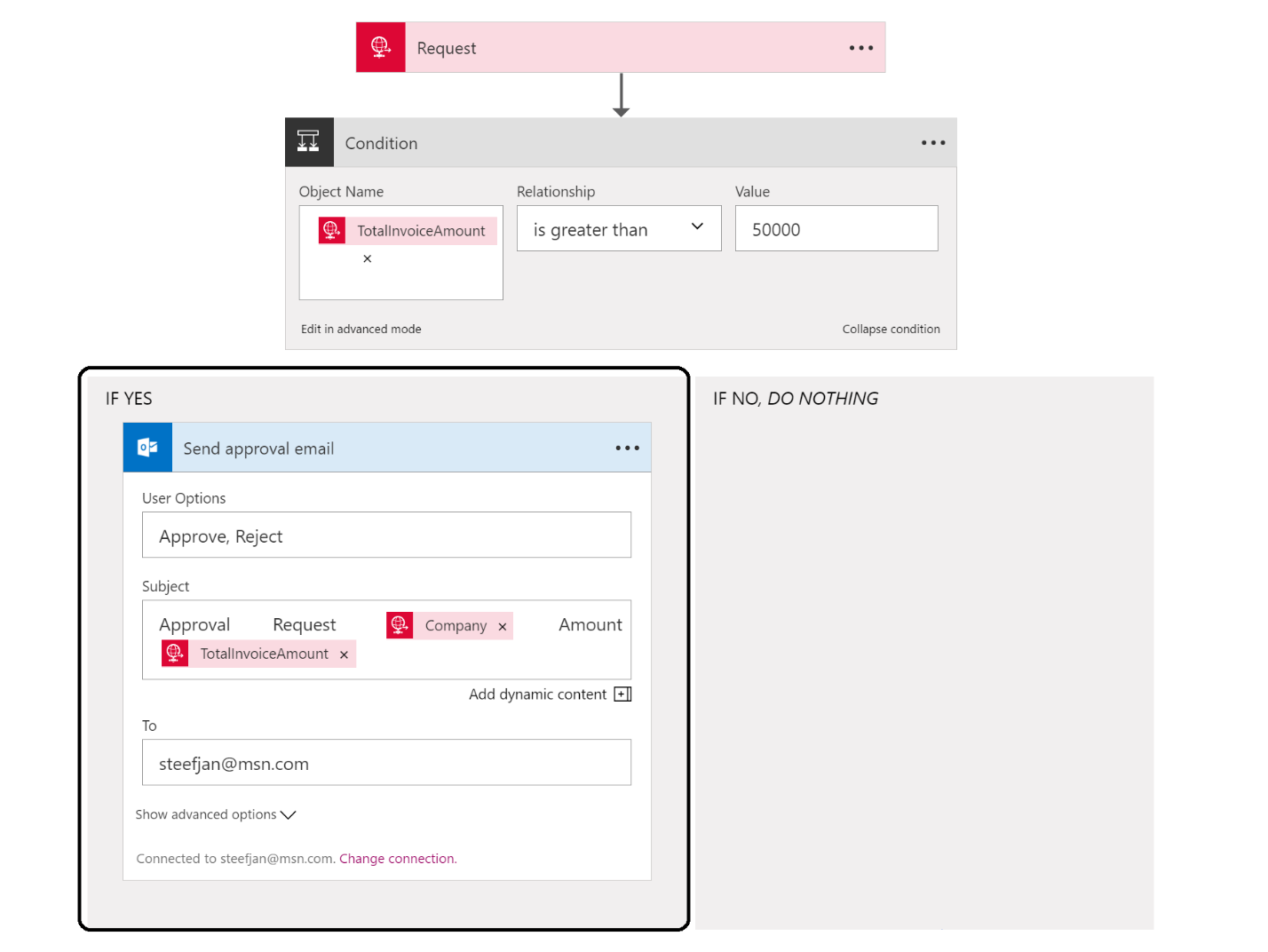
1. The **condition** will be to verify if the amount is higher than **50000**.



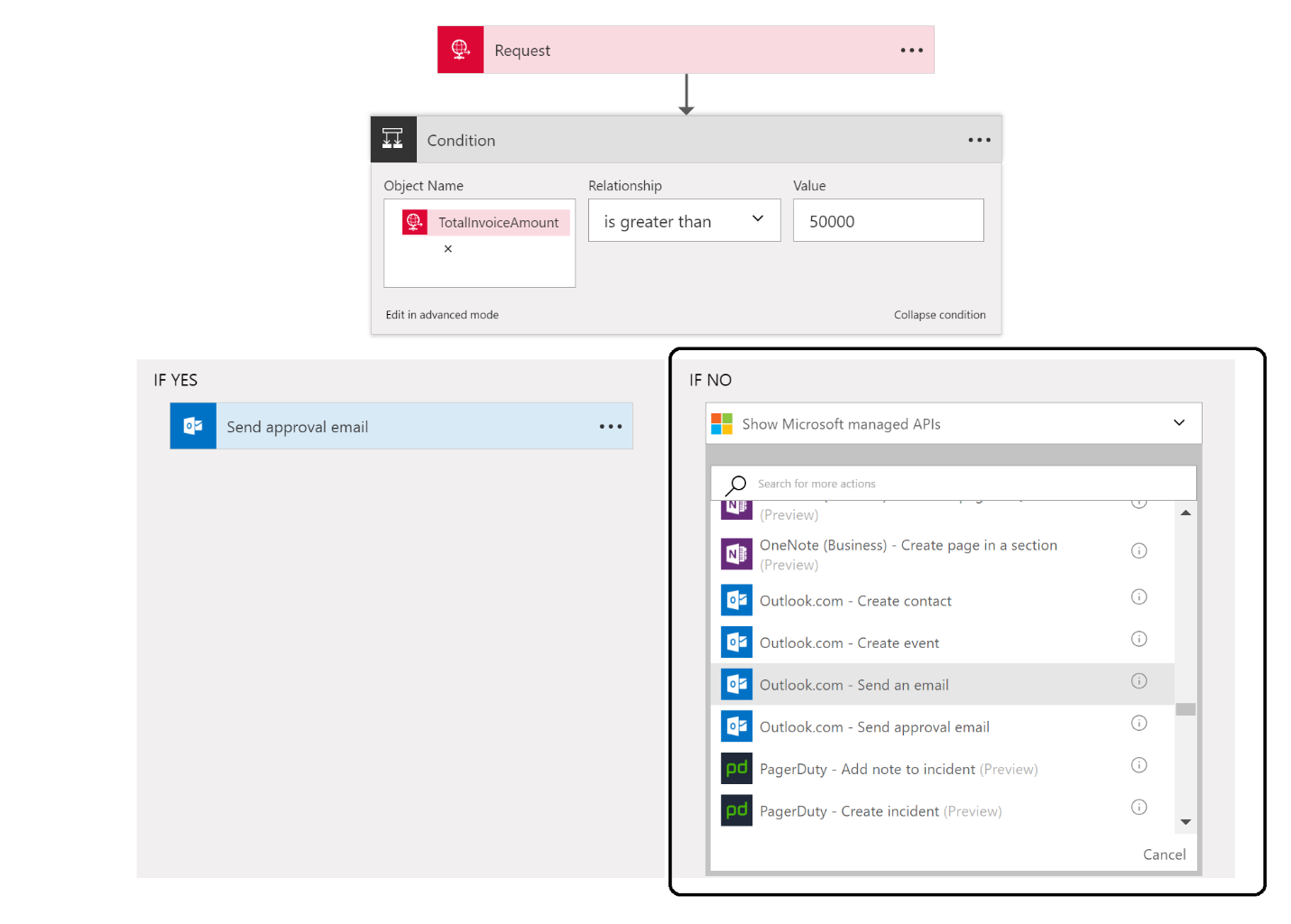
1. Add an **Outlook** (Send Approval Mail) **action** to the left branch.



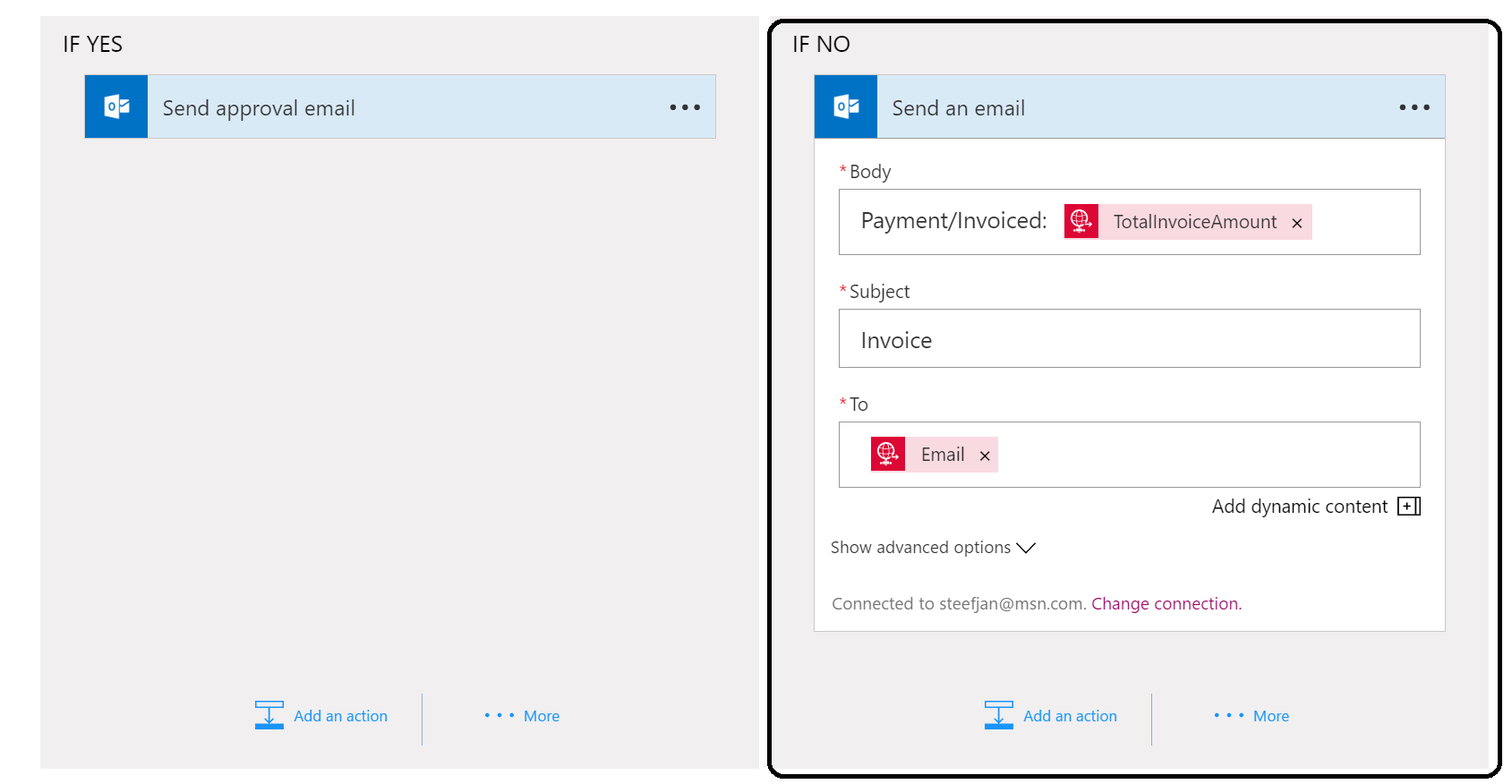
1. **Login** to your **Outlook.com** account. If you do not have one you can create one easily.



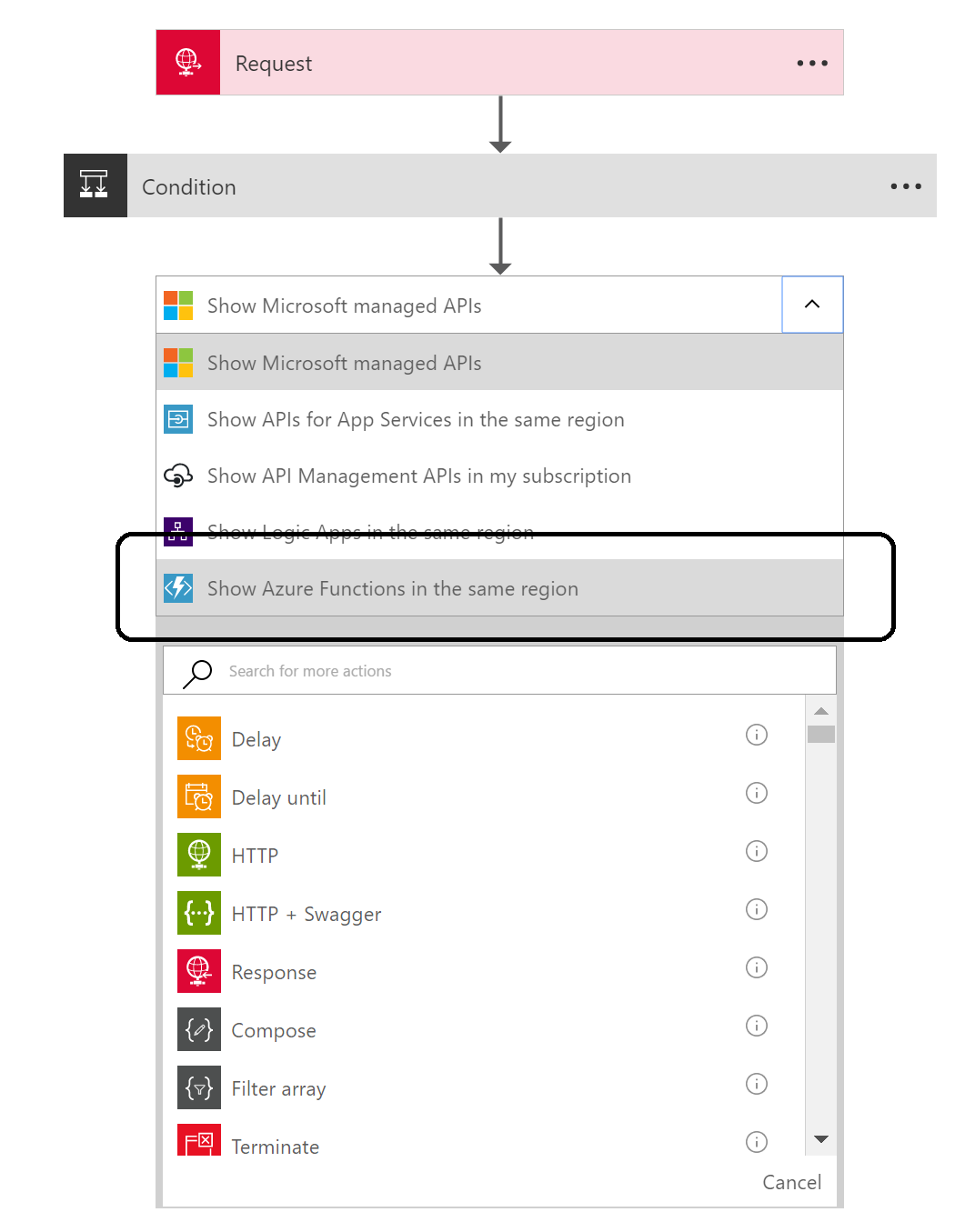
1. After the connection to your outlook has been established you can specify the subject like above and add your email address in the **To** field.



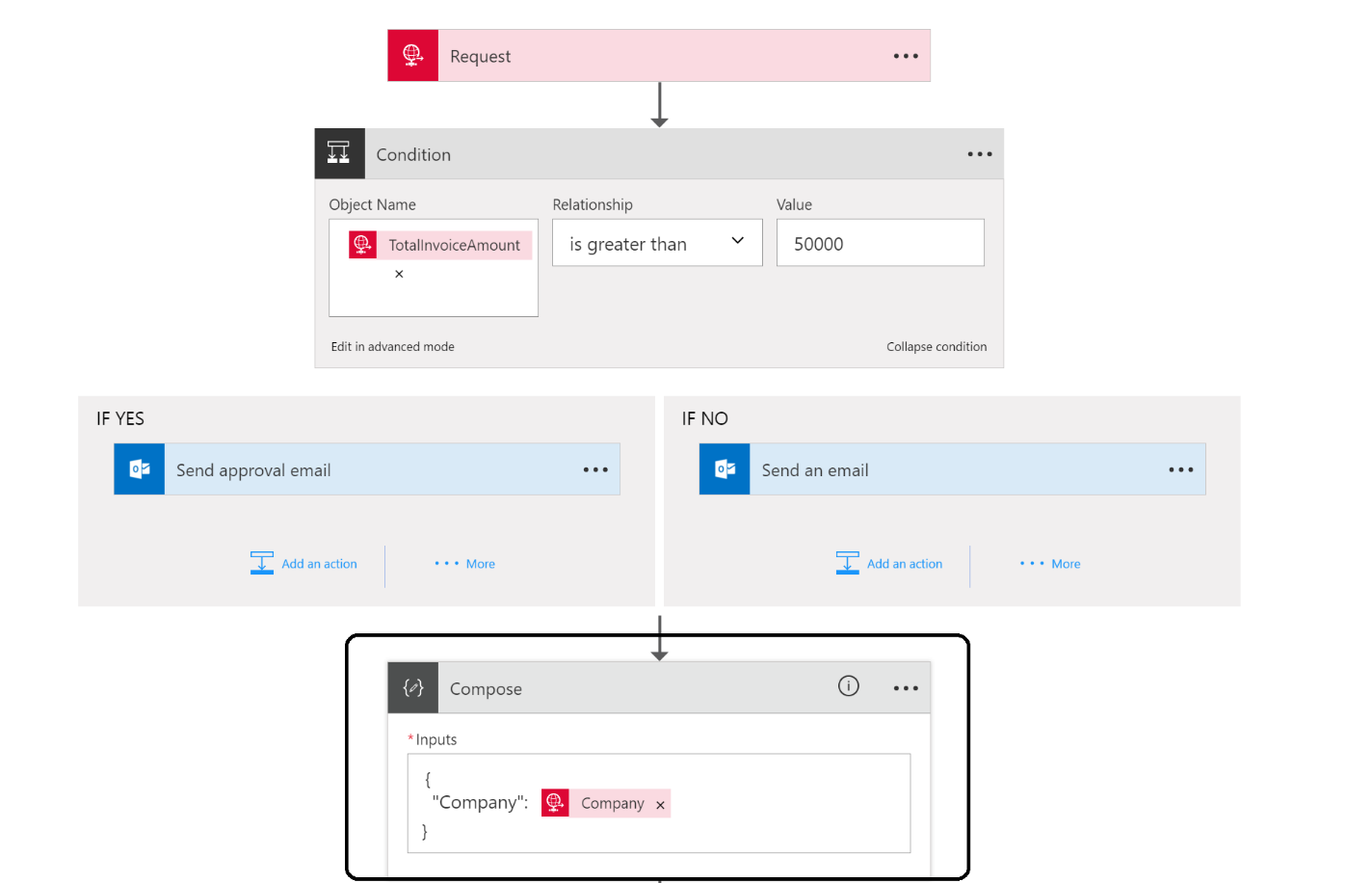
1. In the right branch (IF NO) you can add an Outlook.com Send an email action using your earlier established connection. You do not need to login again.



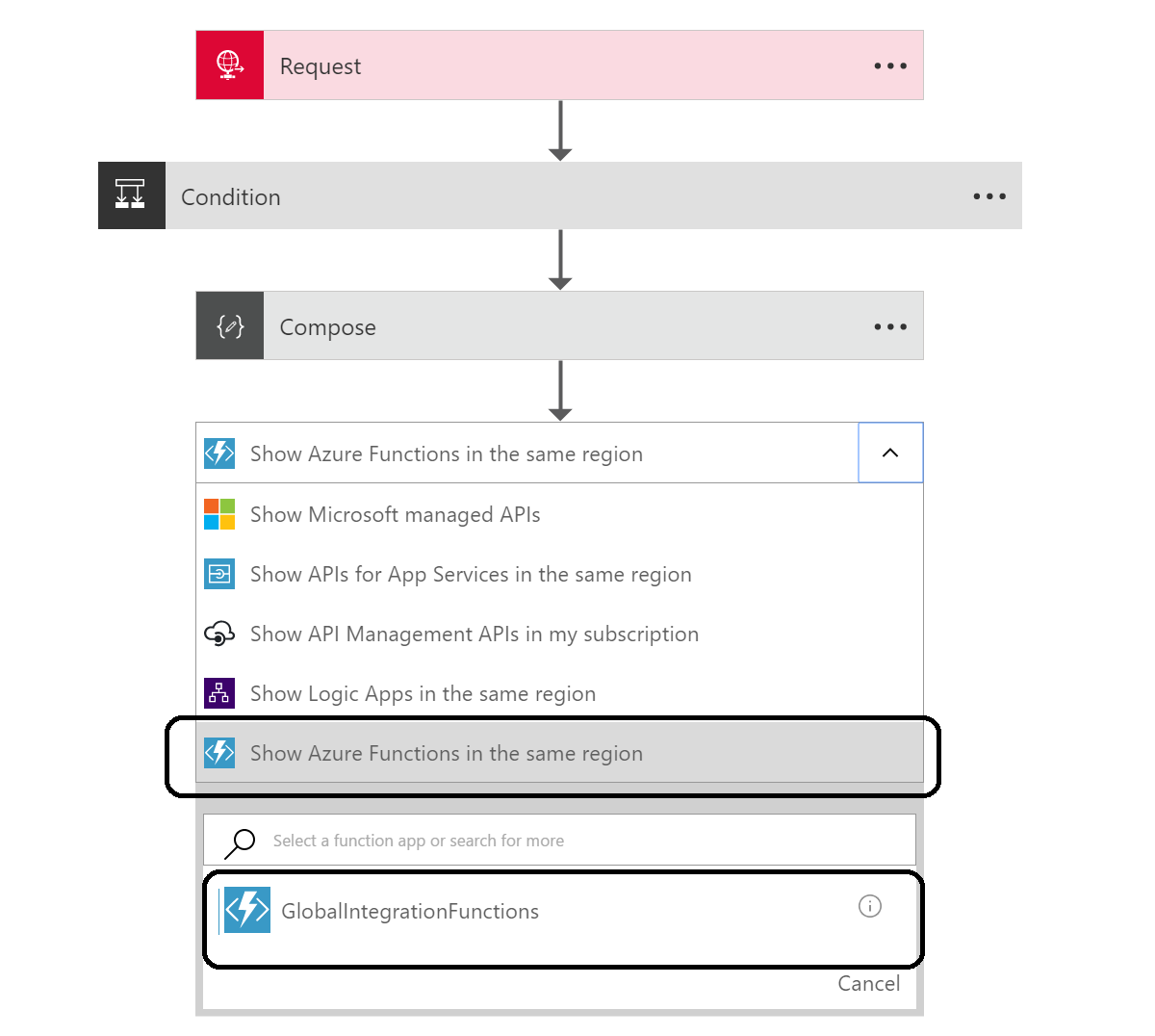
1. You can specify the body as shown above, the subject and the **To** email (which is email to your email address).
2. Add another action below the condition branches
3. Select **Show Azure Functions** in the same region.



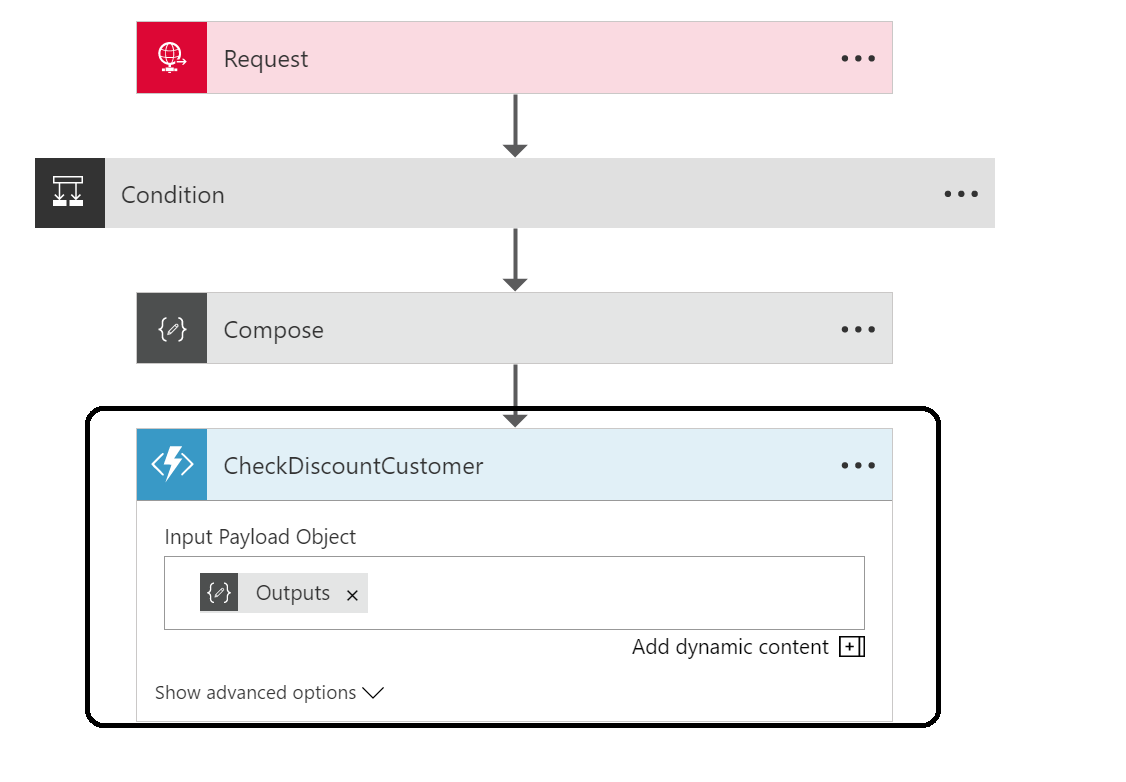
1. Add an **Compose** action below the conditions.



1. In the **Compose** Window create the input as above, making sure to wrap the Company field in quotation marks, otherwise the designer will report an error.
2. Add an action below the **Compose**.
3. Select your function app that will appear.



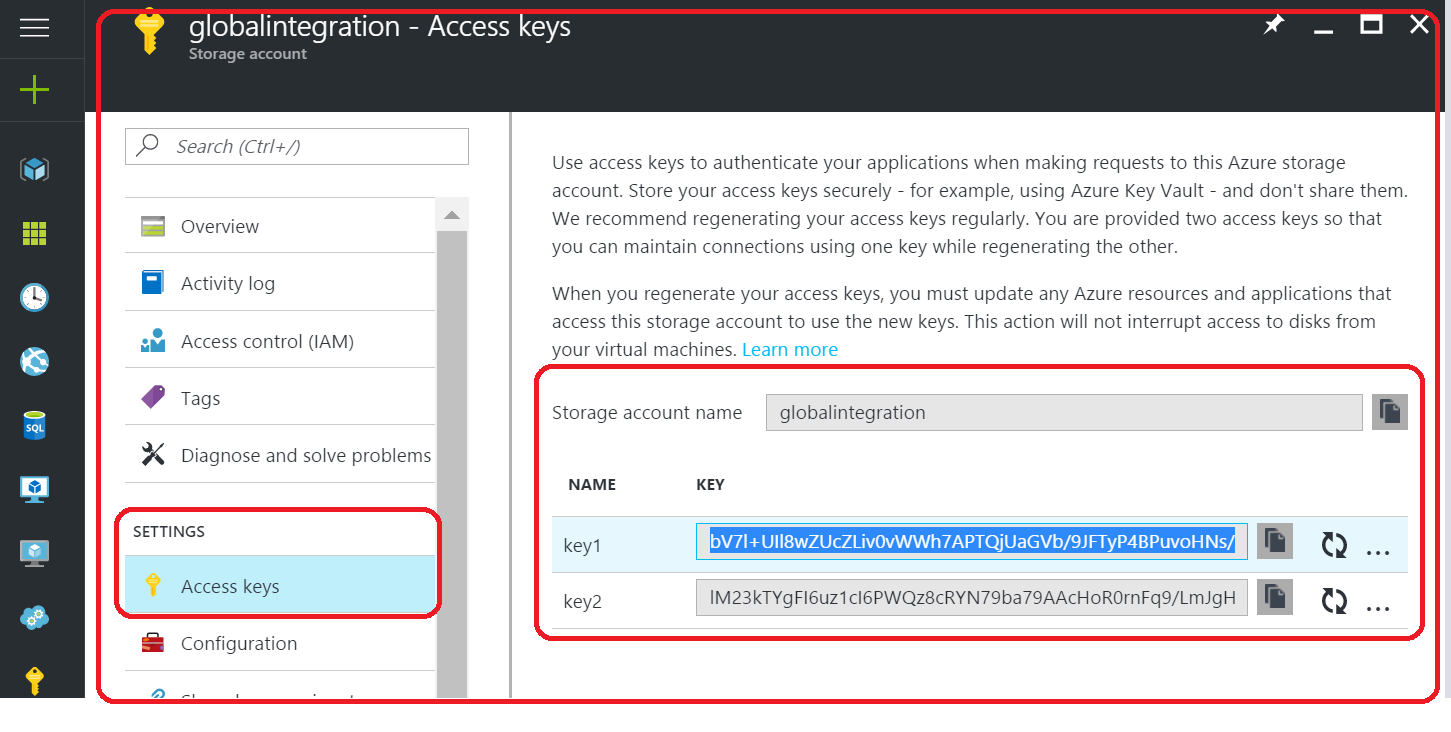
1. Select your custom created function.
2. Drag the **Output** from the **Compose** in the **Input Payload Object**.

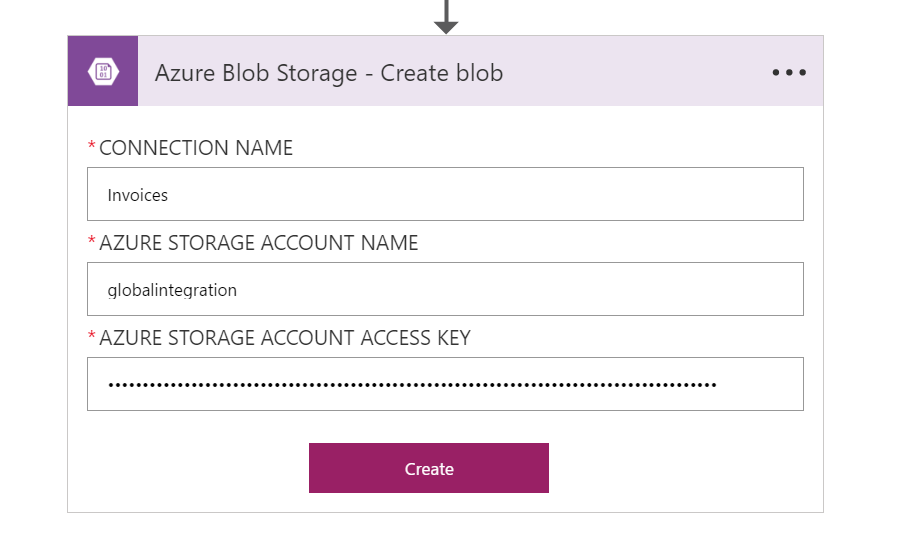


1. Add another action **Azure Blob Storage - Create Blob**.

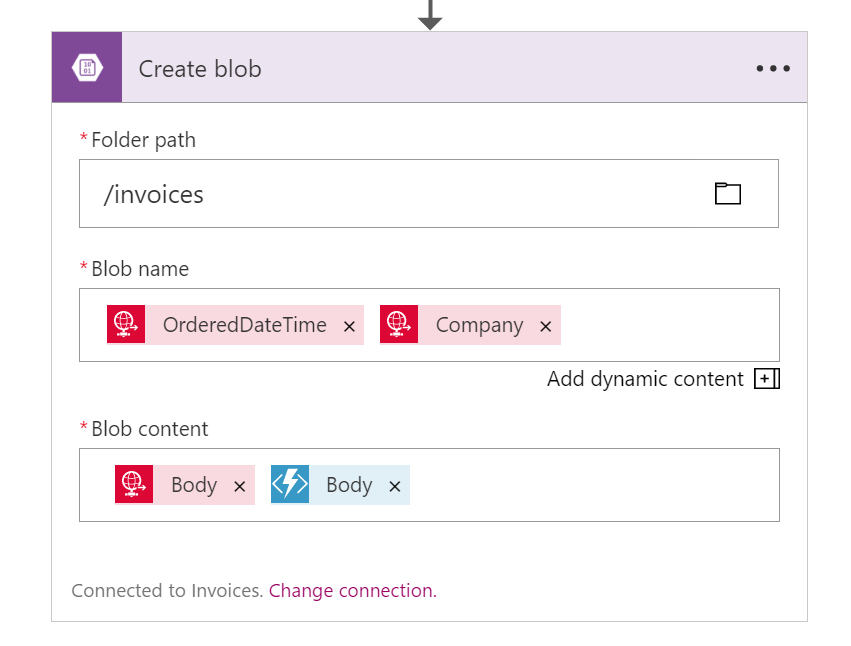


1. Specify the connection **Name**, **Storage** **Account** and **Key**. You can obtain the Account and Key from your previously created storage account.





1. Click **Create**.
2. Now you can select the path i.e. container for the blobs (invoices), blob name and define the content.

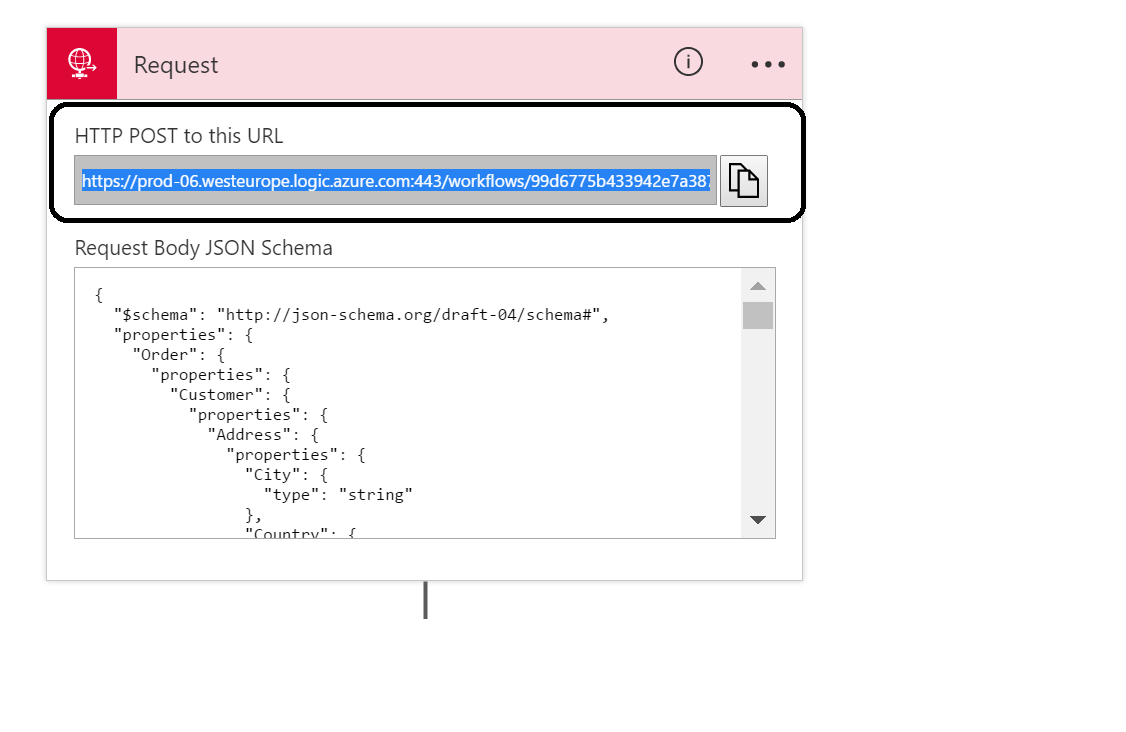


1. **Save** the Logic App Definition.

## Test the Solution

Our solution has completely setup and can be tested now. And to test the solution you can use Postman, which can be obtained through online.

1. To be able to post anything to the **Logic App** you’ll need to open up the **Logic App Designer** and copy the **URL**.



1. Open **postman**.
2. Copy address in address bar, select **POST** Method, for the body choose **raw** and content type **application/json**.
3. Post the following as body content i.e. you can change **TotalInvoiceAmount** to an amount below 50000, and specify your own email address!

**{**

"Order"**:{**

"Customer"**:{**

"Company"**:**"Macaw"**,**

"Email"**:**"steefjan@msn.com"**,**

"CustomerNumber"**:**"6f6d4907-23af-e611-80e5-5065f38a5a01"**,**

"Address"**:{**

"Street"**:**"Beechavenue 140"**,**

"City"**:**"Schiphol-Rijk"**,**

"PostalCode"**:**"1119 PR"**,**

"Country"**:**"Netherlands"

**}**

**},**

"Products"**:{**

"Product"**:[**

**{**

"ProductNumber"**:**1000**,**

"Amount"**:**1**,**

"Price"**:**123.45

**},**

**{**

"ProductNumber"**:**2000**,**

"Amount"**:**5**,**

"Price"**:**456.78

**}**

**]**

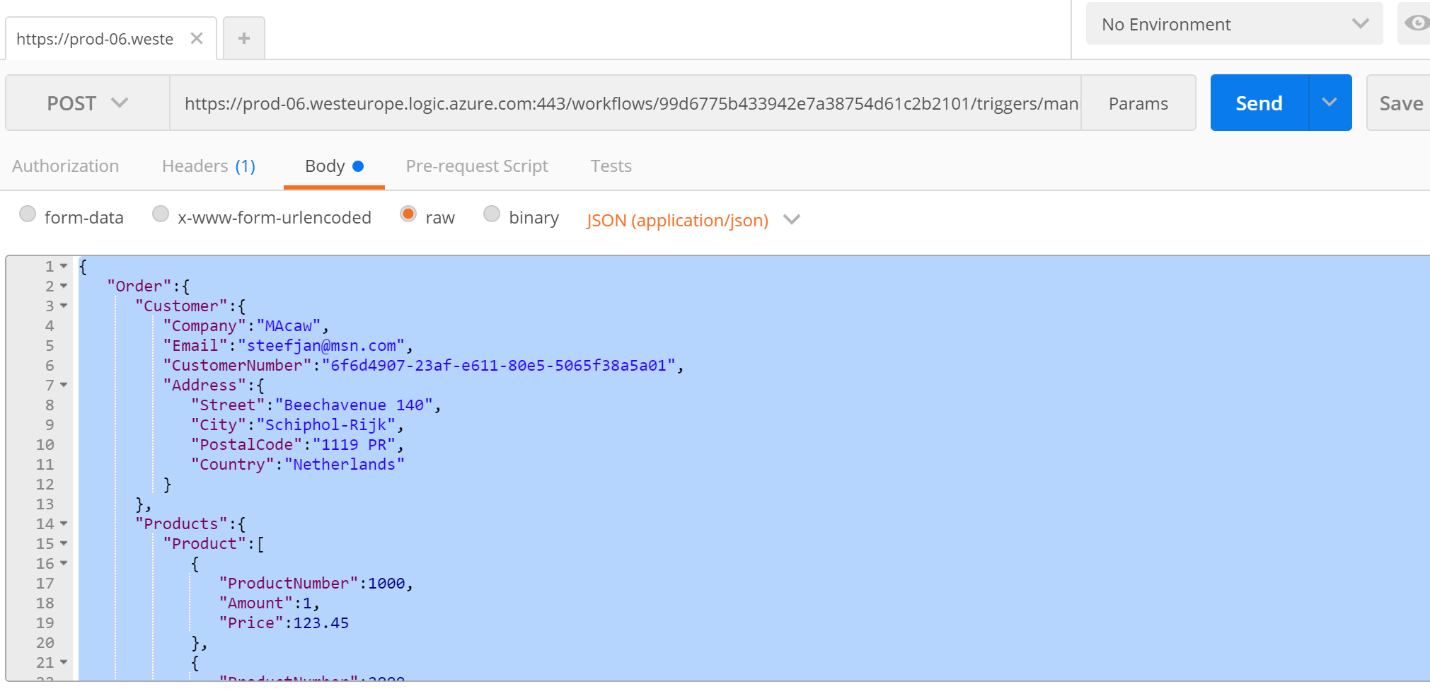
**},**

"OrderedDateTime"**:**"2016-11-20T14:26:00"**,**

"TotalInvoiceAmount"**:**2407.35

**}**

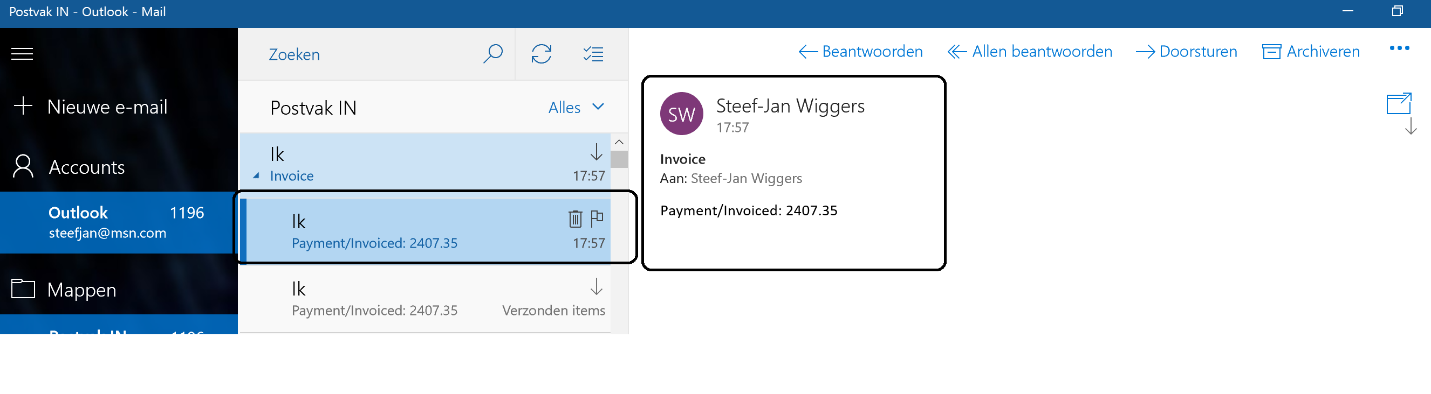
**}**

****

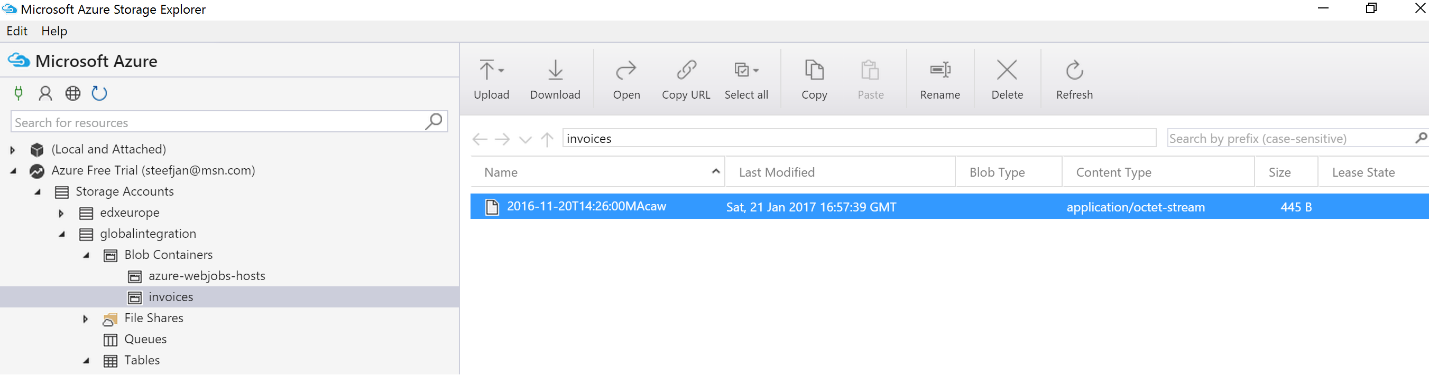
1. Click **Send**.
2. Switch over to **Azure Portal**.
3. Go to your **Logic App**.
4. Click on the last run and examine the steps.



1. Look into your **mail box** and see if you have an **email**.



1. Check using the **Azure Explorer** if there’s a blob in the invoices container.



1. **Click** on the blob and examine the contents and look for the discount.
2. Repeat the test with an amount above **50000**.
3. You’ll have to look into your mailbox to see an approval mail, which you can approve. Please note that the HTTP trigger has a time-out, so if this takes too long, the Logic App can get into a timeout. Using the Service Bus trigger this will not be an issue.