

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](#_Toc480532928)

[Logic Apps and Azure Lab 4](#_Toc480532929)

[Scenario 4](#_Toc480532930)

[Prerequisites 4](#_Toc480532931)

[Steps 5](#_Toc480532932)

[Provison the Storage Account 6](#_Toc480532933)

[Create Strorage Container 8](#_Toc480532934)

[Create Storage Table 9](#_Toc480532935)

[Provision the Function App 12](#_Toc480532936)

[Building a Function 13](#_Toc480532937)

[Provision a Logic App 18](#_Toc480532938)

[Building a Logic App Definition 20](#_Toc480532939)

[Test the Solution 29](#_Toc480532940)

[Creating a Proxy Function 33](#_Toc480532941)

[Enabling Proxy Functionality 33](#_Toc480532942)

[Creating a Proxy Function 34](#_Toc480532943)

[Testing the Proxy Function 36](#_Toc480532944)

# 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-processorder-<ini> | gab2017akl-logic-processorder-ws |
| 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

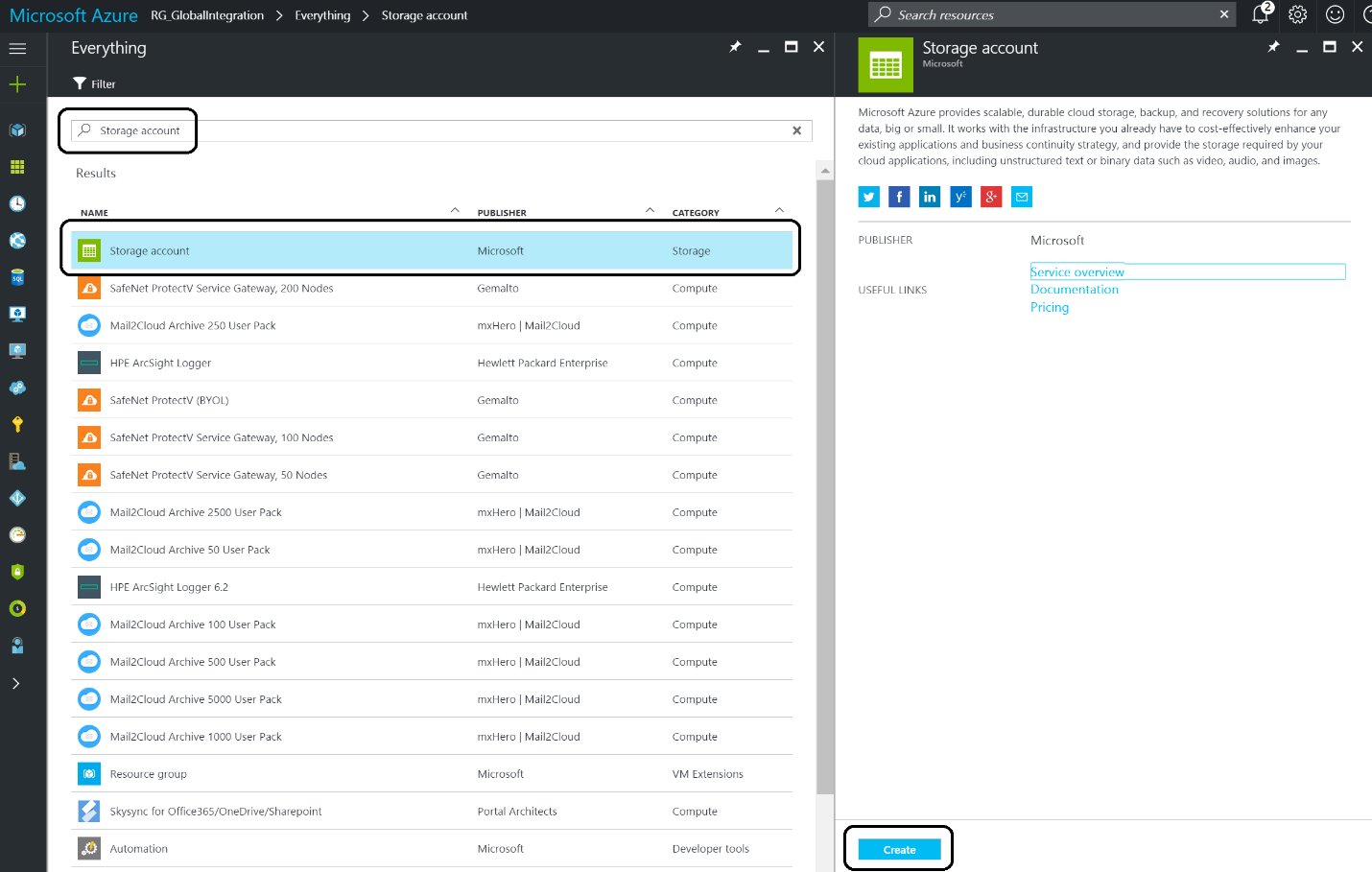
To build the solution in this lab you have to follow the steps described in this section. From a high level view the steps are:

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

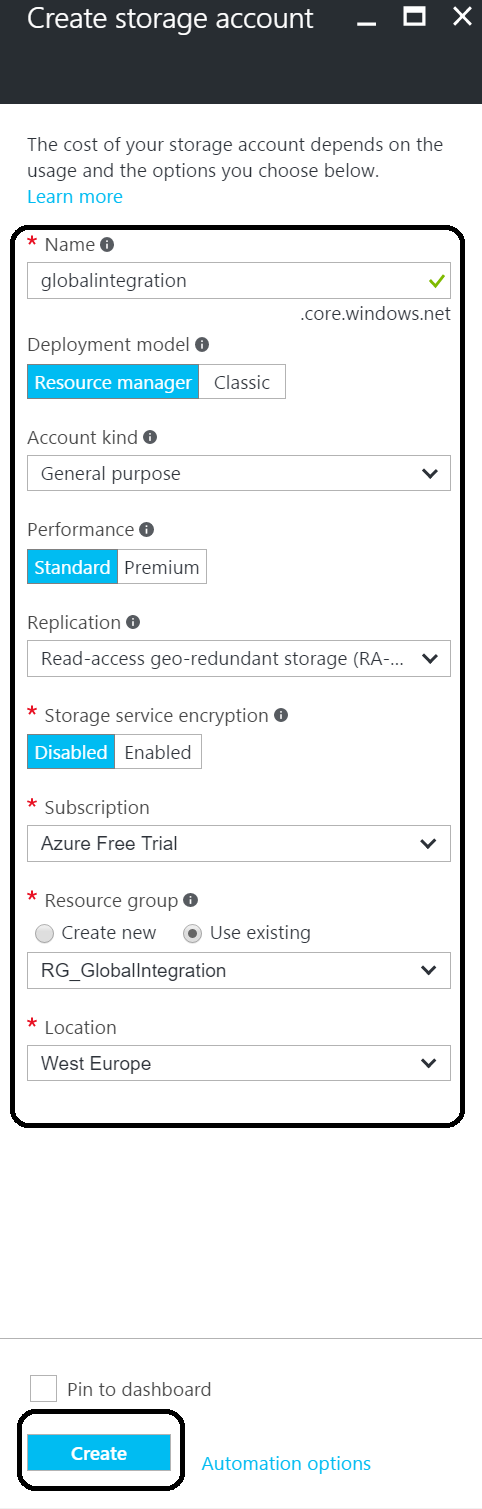
## Provison the Storage Account

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**.

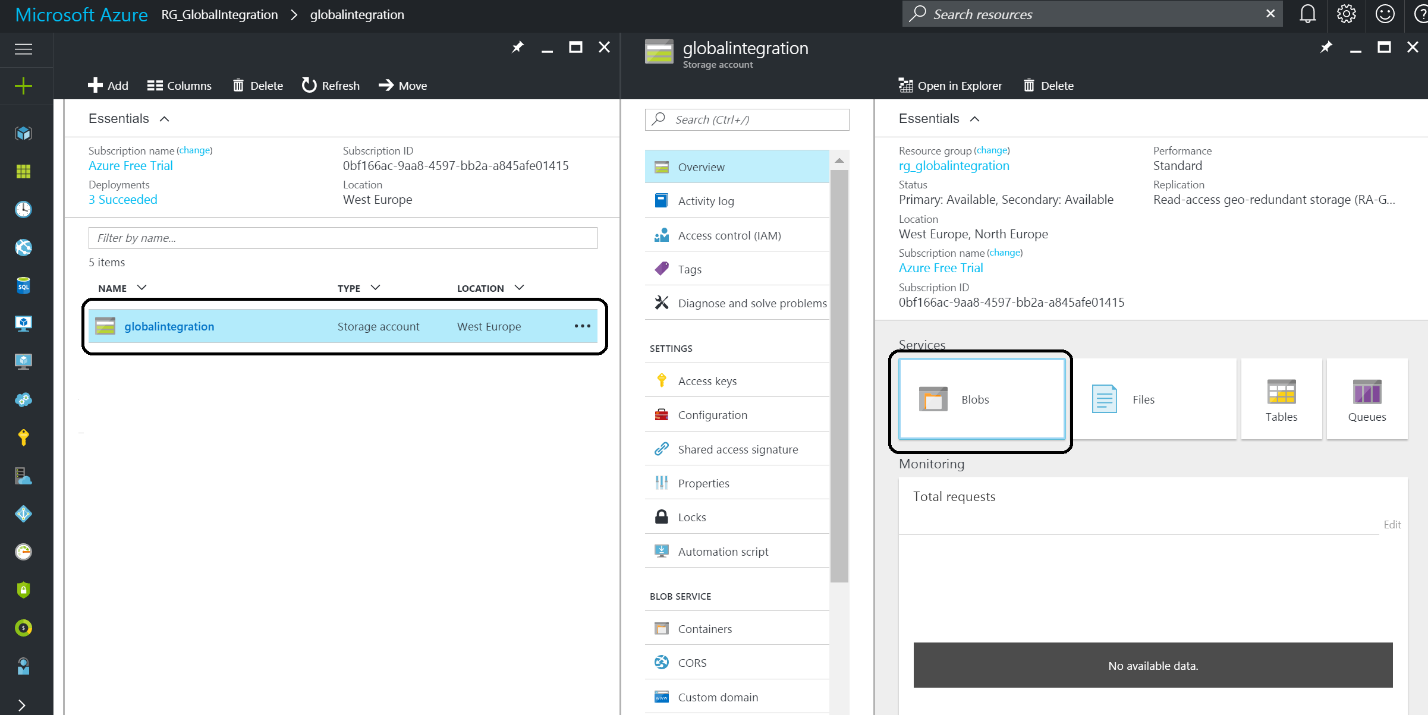


1. Specify the name- **gab2017aklstor<ini>**, the **Resource Group – gab2017akl-rg-<ini>** (you can create a new one here if you haven’t created a resource group yet) and a **location**. Subsequently, click on **Create**.

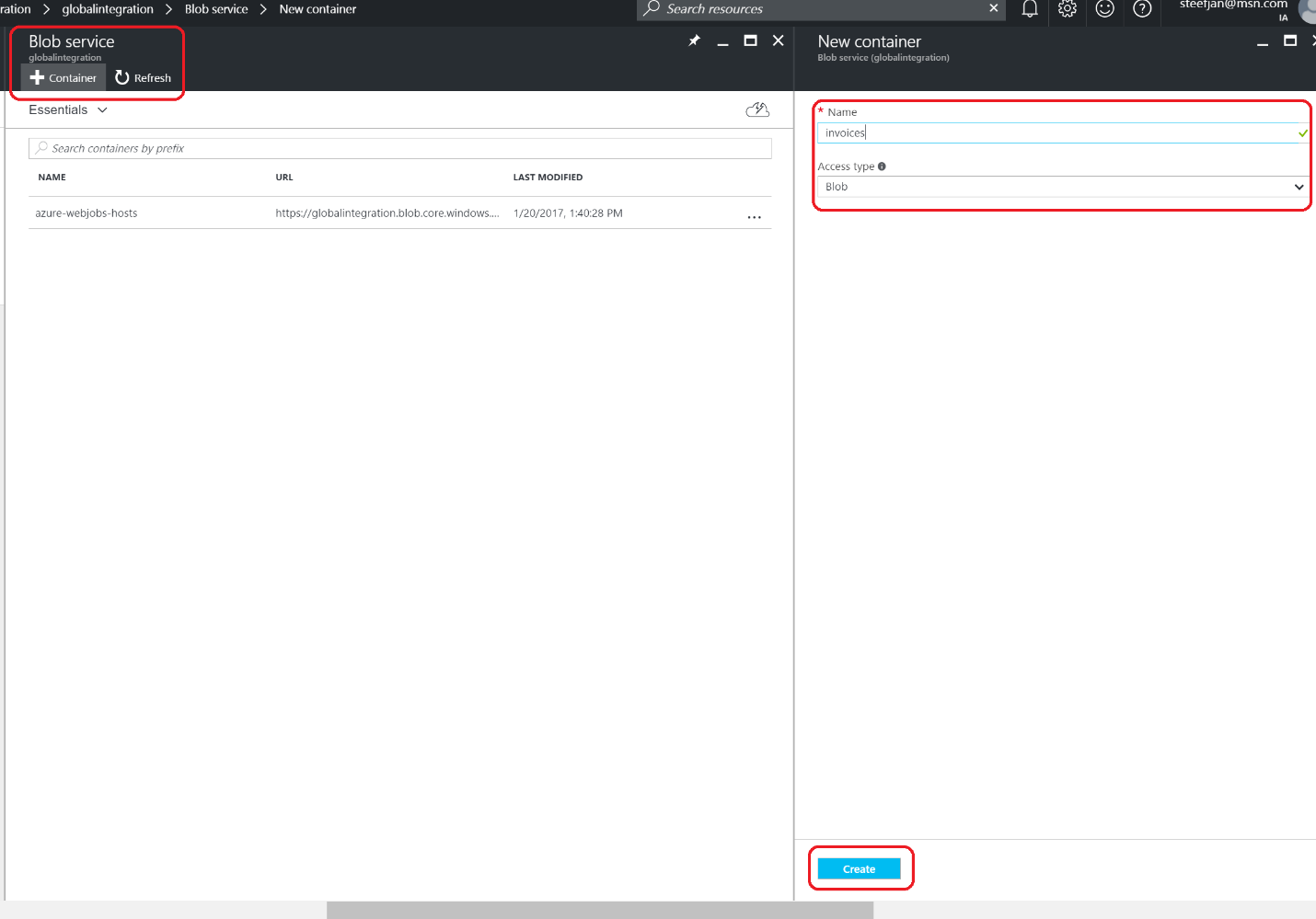
## Create Strorage Container

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

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



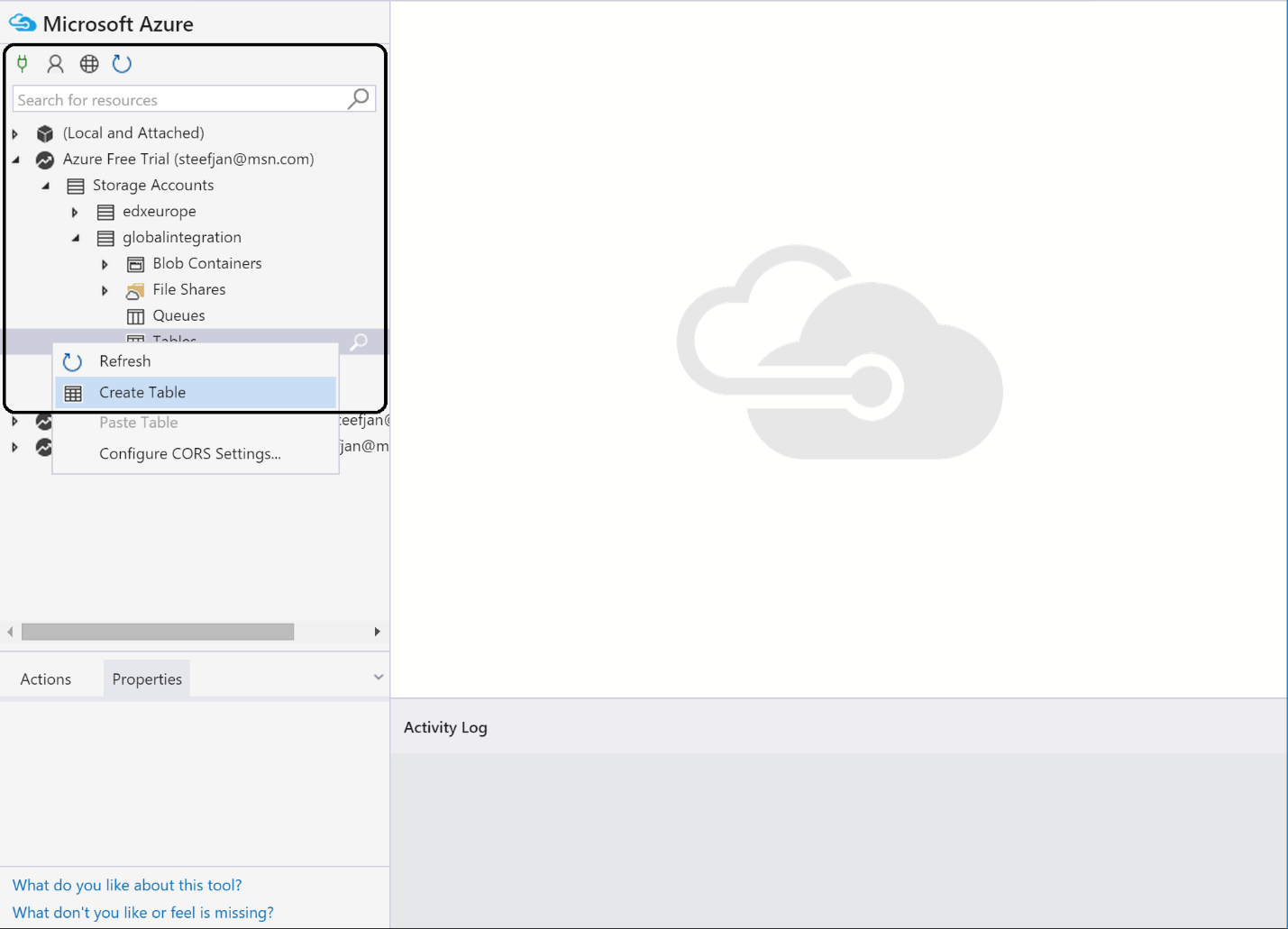
1. Click on **+ Container** and specify the name – **invoices -** and Access Type- **Blob**.



## Create Storage Table

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

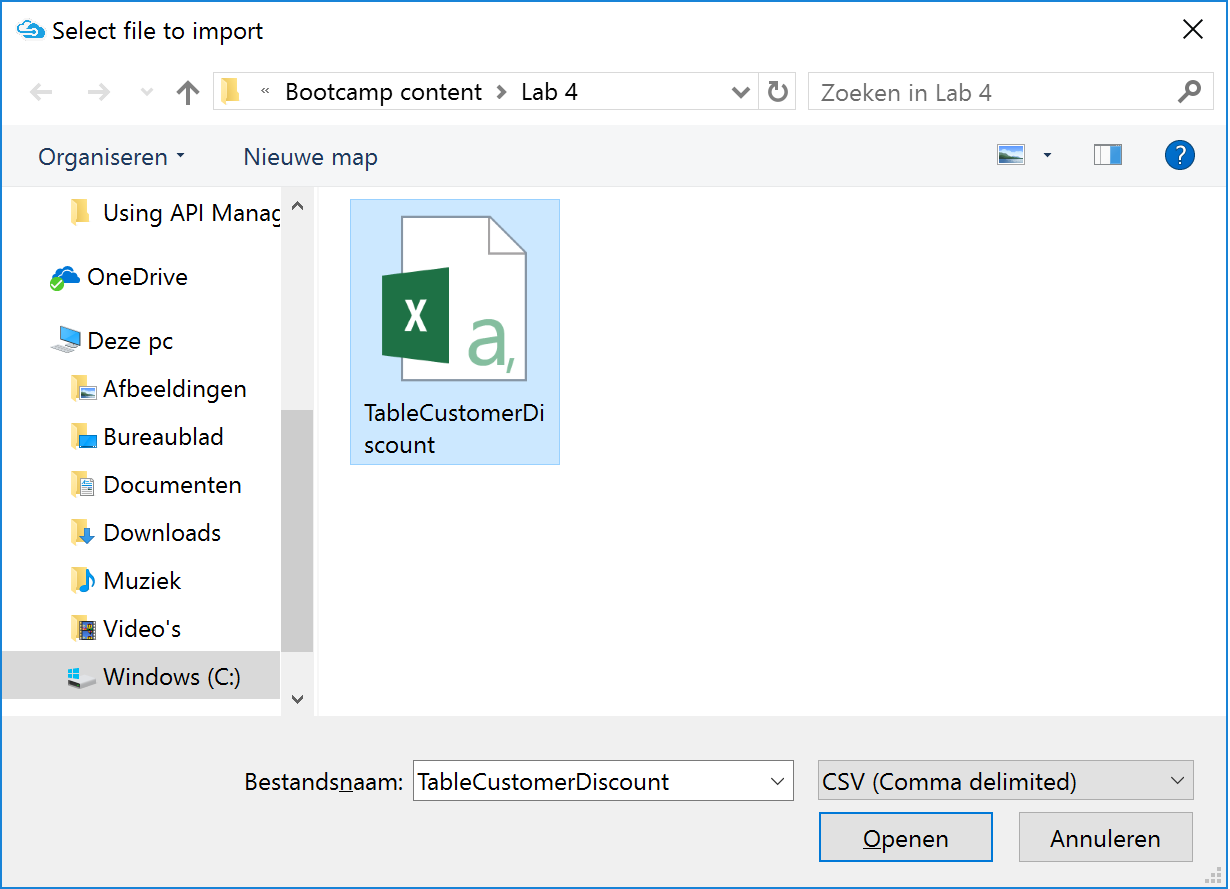
1. Install the tool, and login into your subscription.
2. Navigate to your storage account.
3. Select Tables
4. Right click Tables and click Create Table.



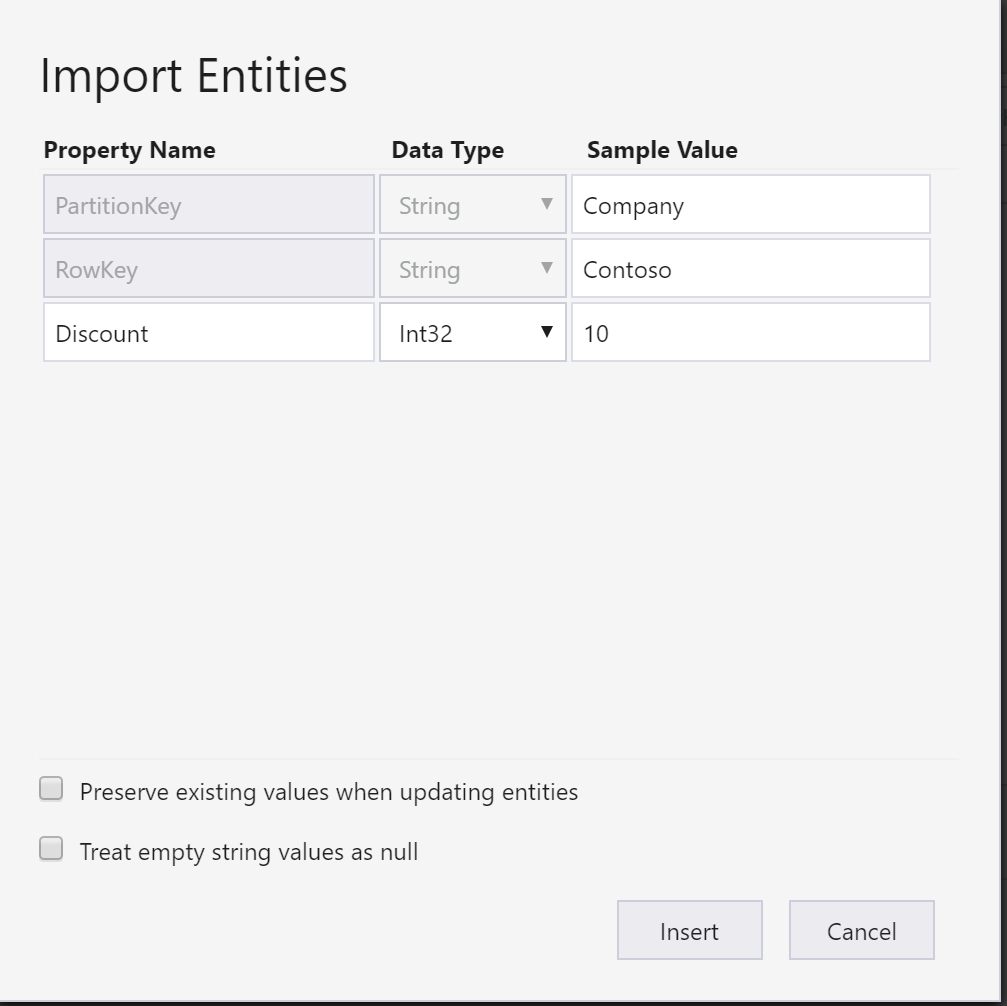
1. Specify a name for the table - **CustomerDiscount**.
2. Select the **table**.
3. Click on **Import** **Entities** from file.



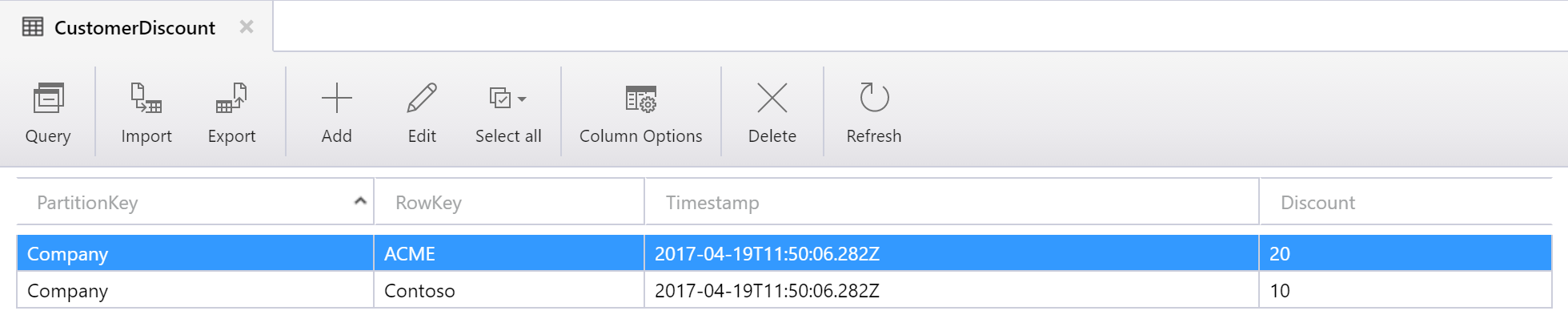
1. Select the **TableCustomerDiscount.csv** you downloaded previously.



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



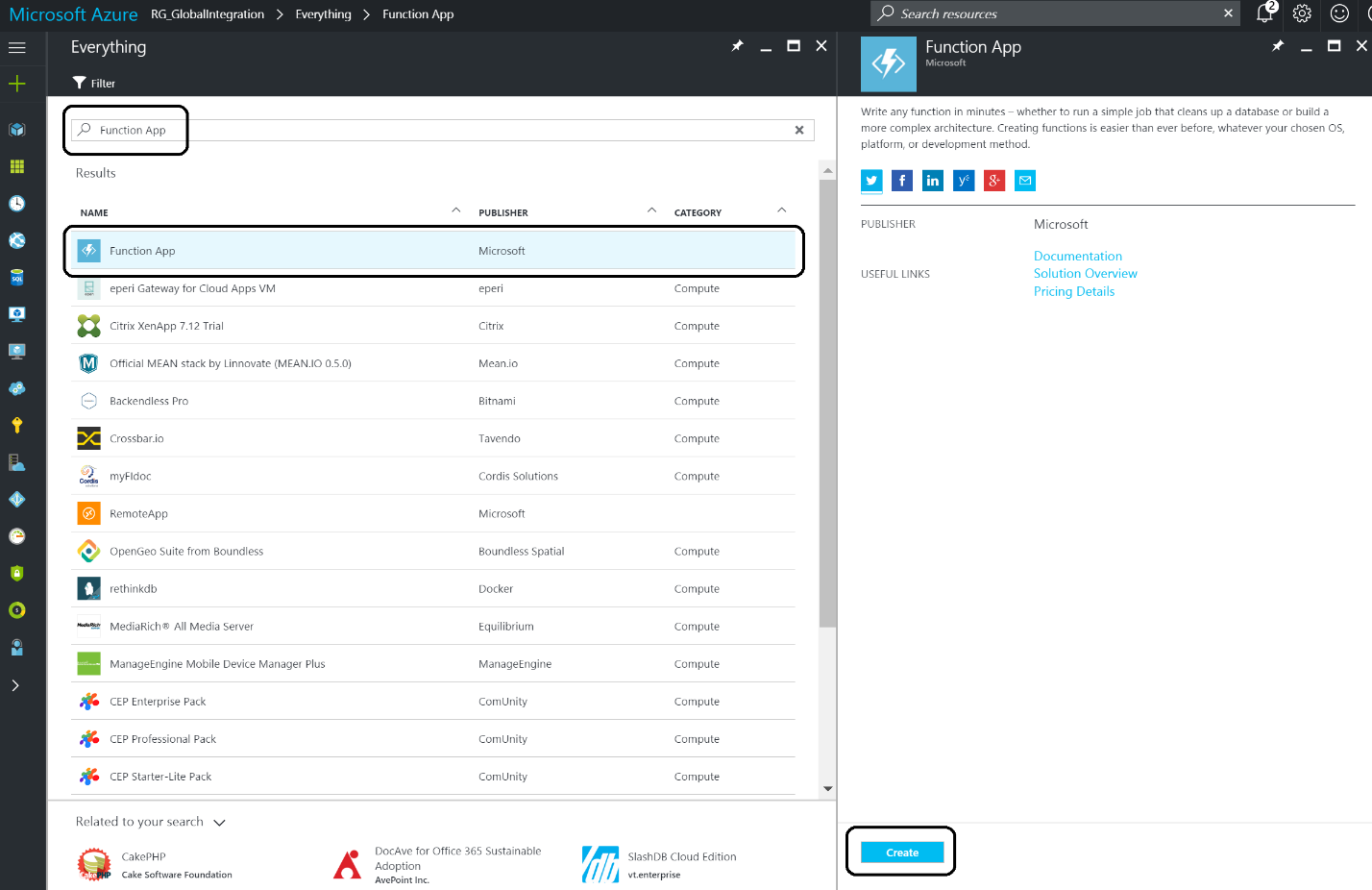
1. **Verify the Table** was loaded with data.

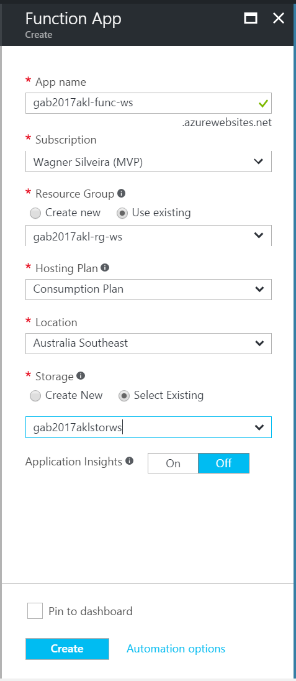


## Provision the Function App

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**.



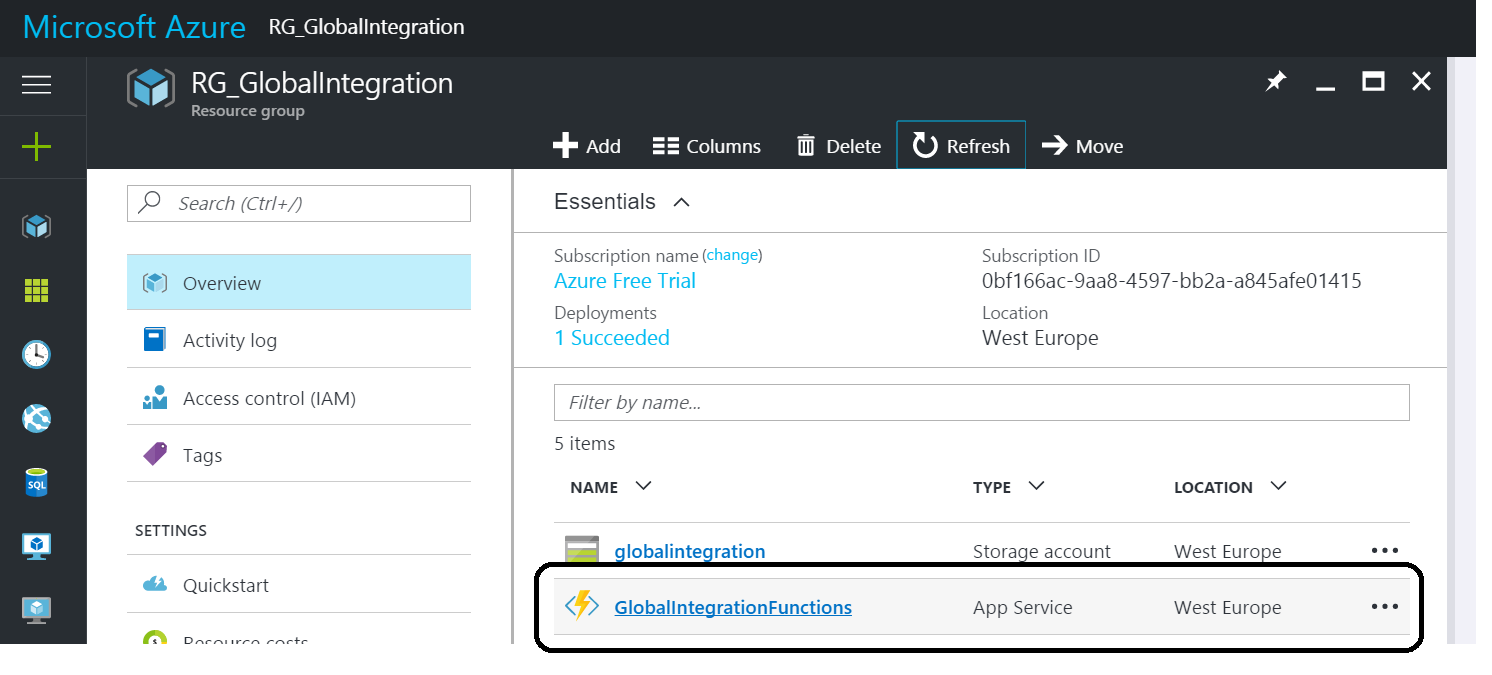


1. Specify the following details:
   1. Name: **gab2017akl-func-<ini>**
   2. Resource Group – select existing: **gab2017akl-rg-<ini>**
   3. Hosting Plan: **Consumption plan**
   4. Location: <will be auto populated by the Resource Group>
   5. Storage – select existing: **gab2017aklstor<ini>**

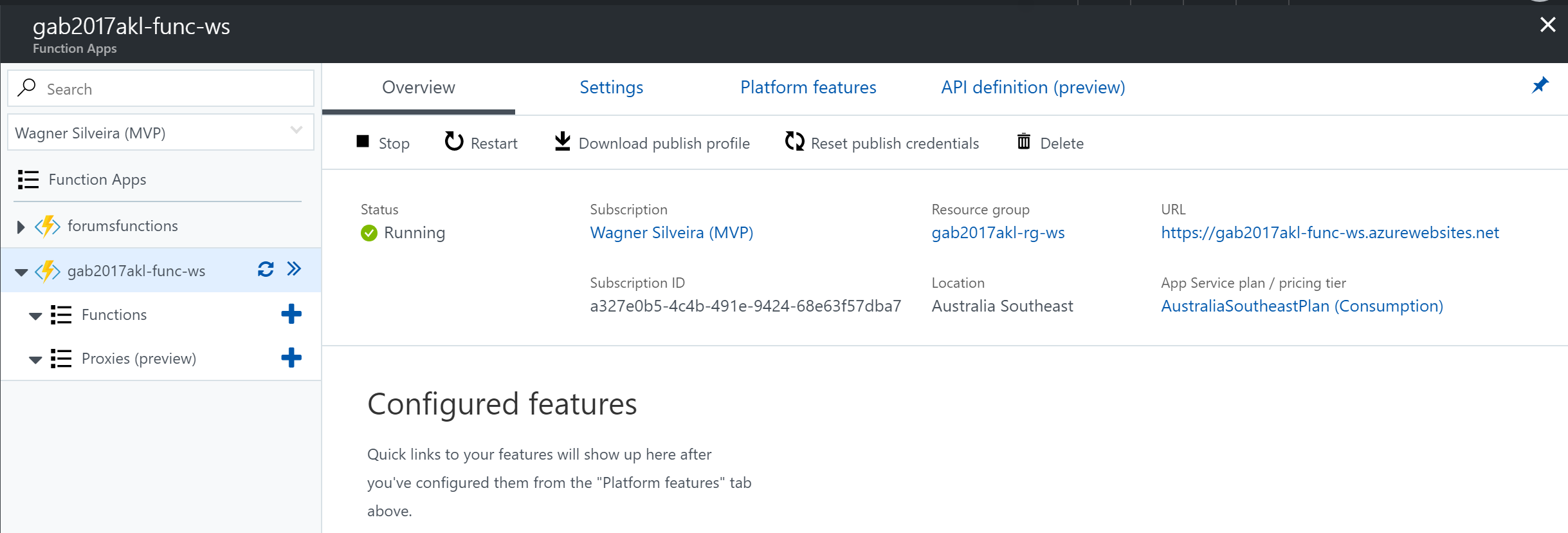
### Building a Function

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

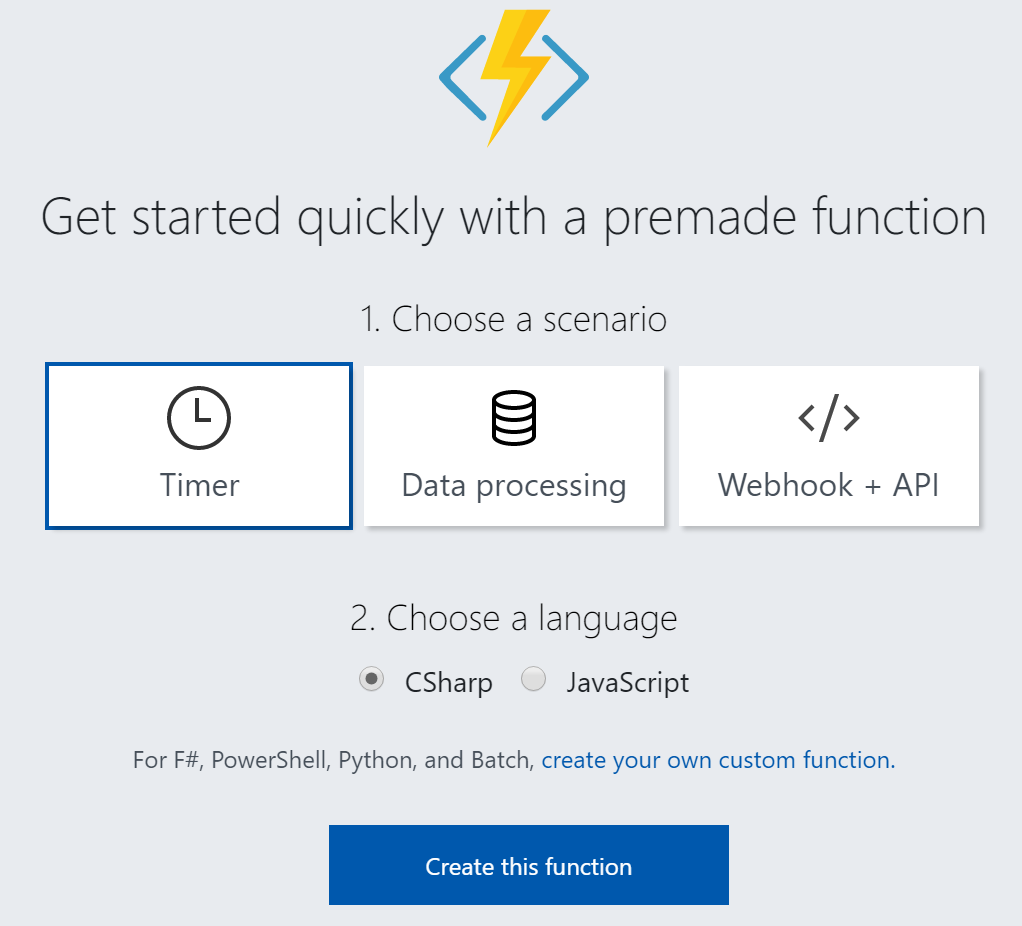
1. Navigate to the **Function App** from the Resource Group:



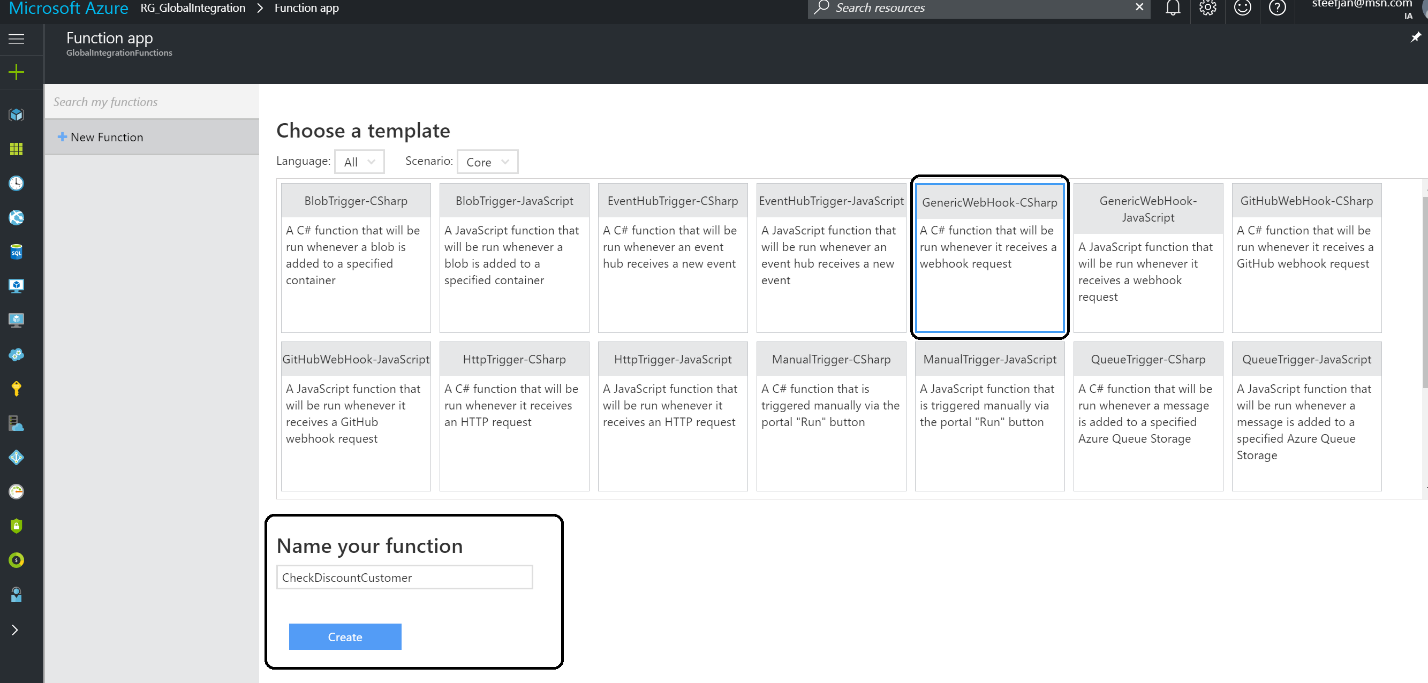
1. In the function app click **Functions +**.



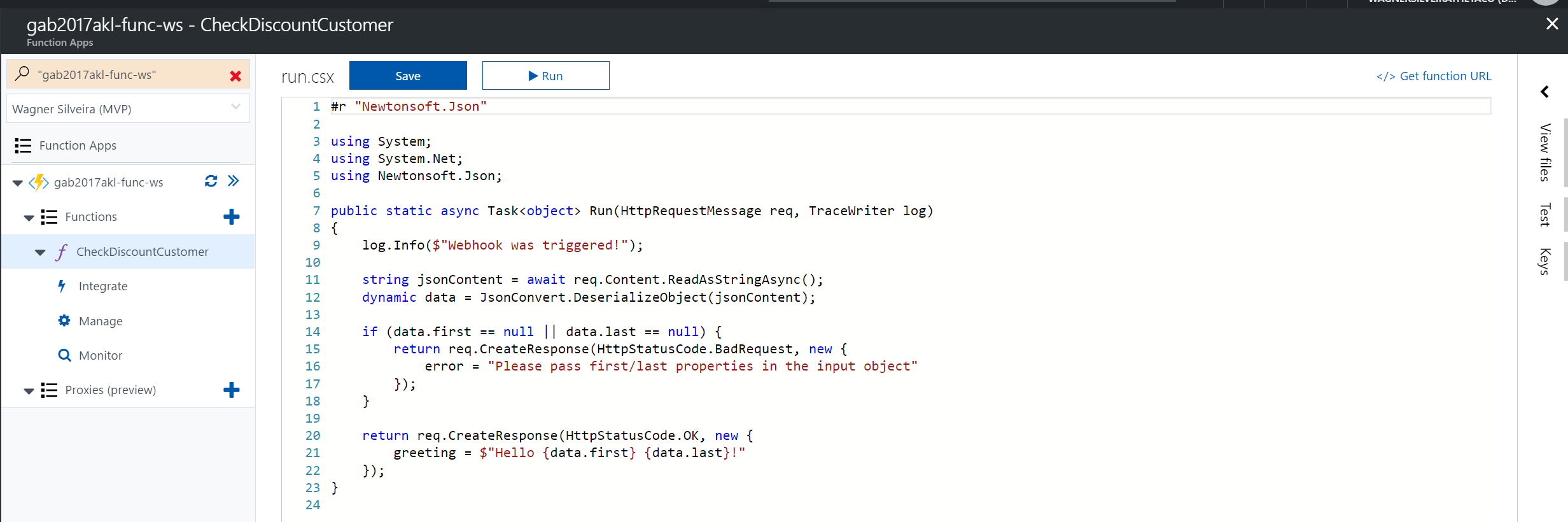
1. In the quick start function, click on **create your own custom function**:



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**);**

**}**

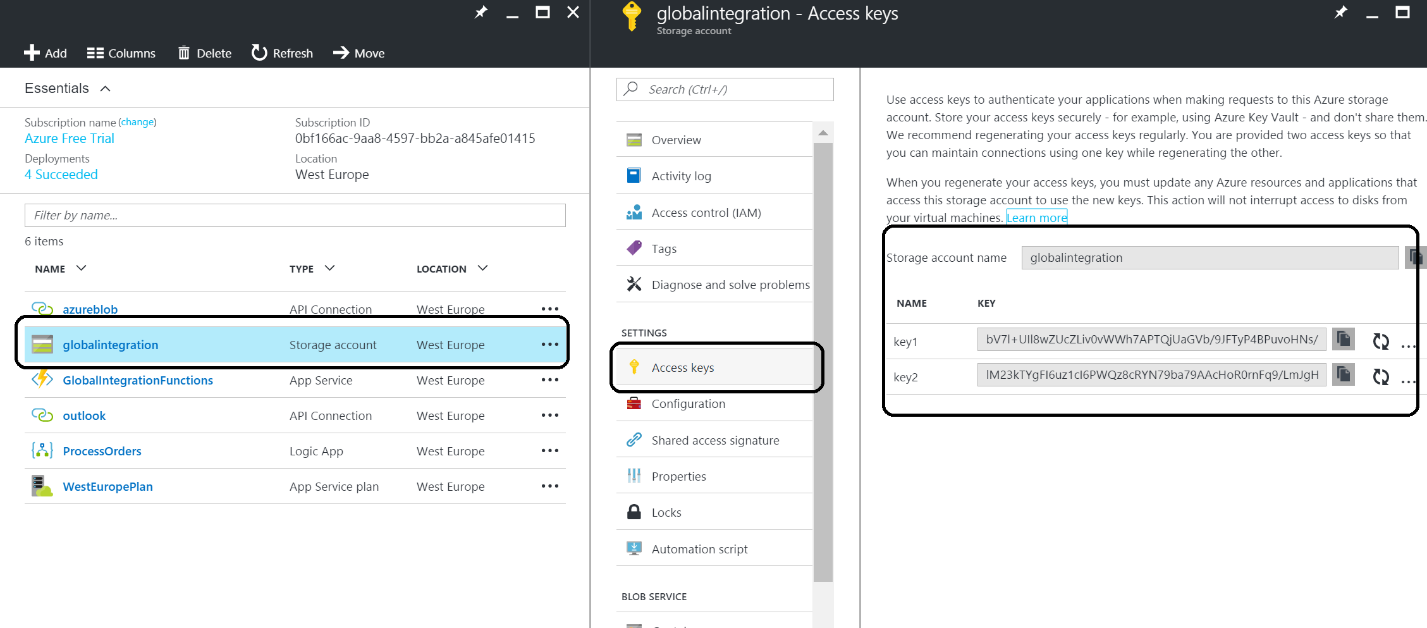
**return** req**.**CreateResponse**(**HttpStatusCode**.**OK**,** **new** **{**

discountValue **=** discount

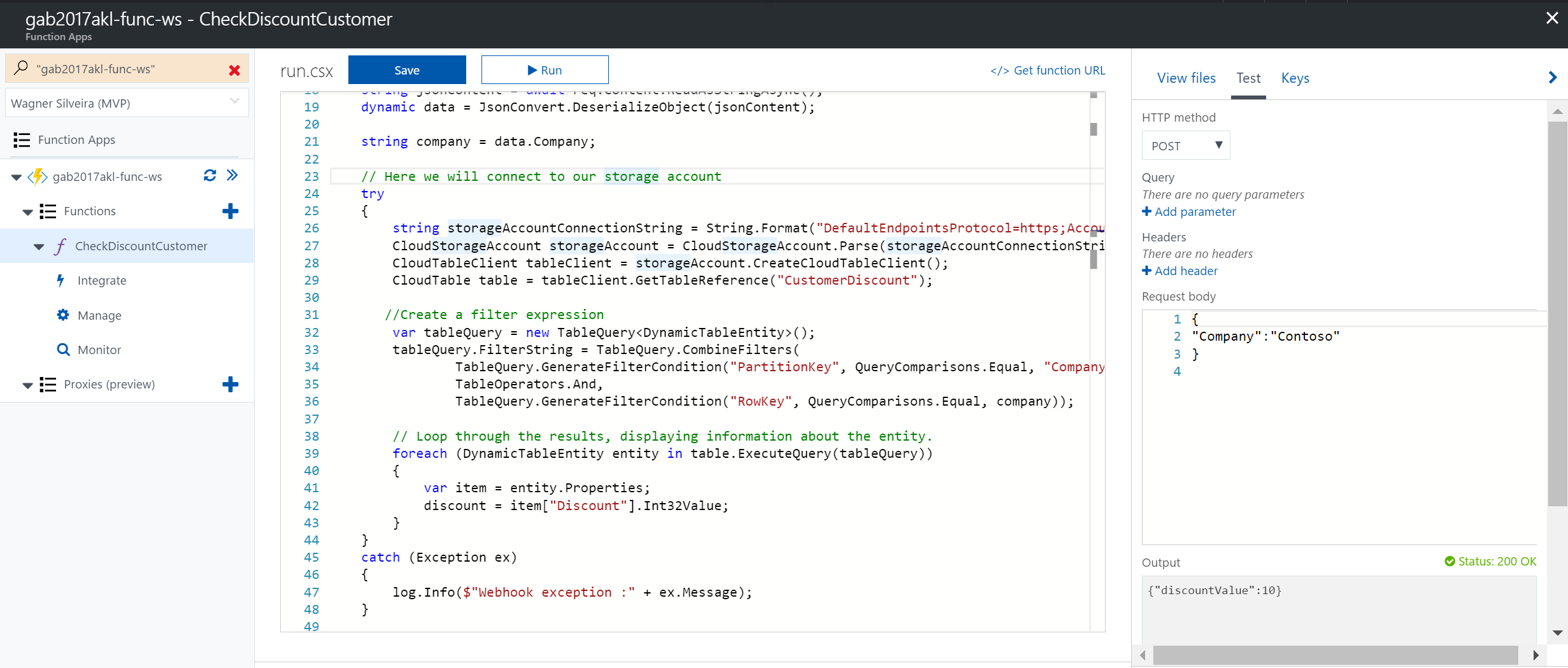
**});**

**}**

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.

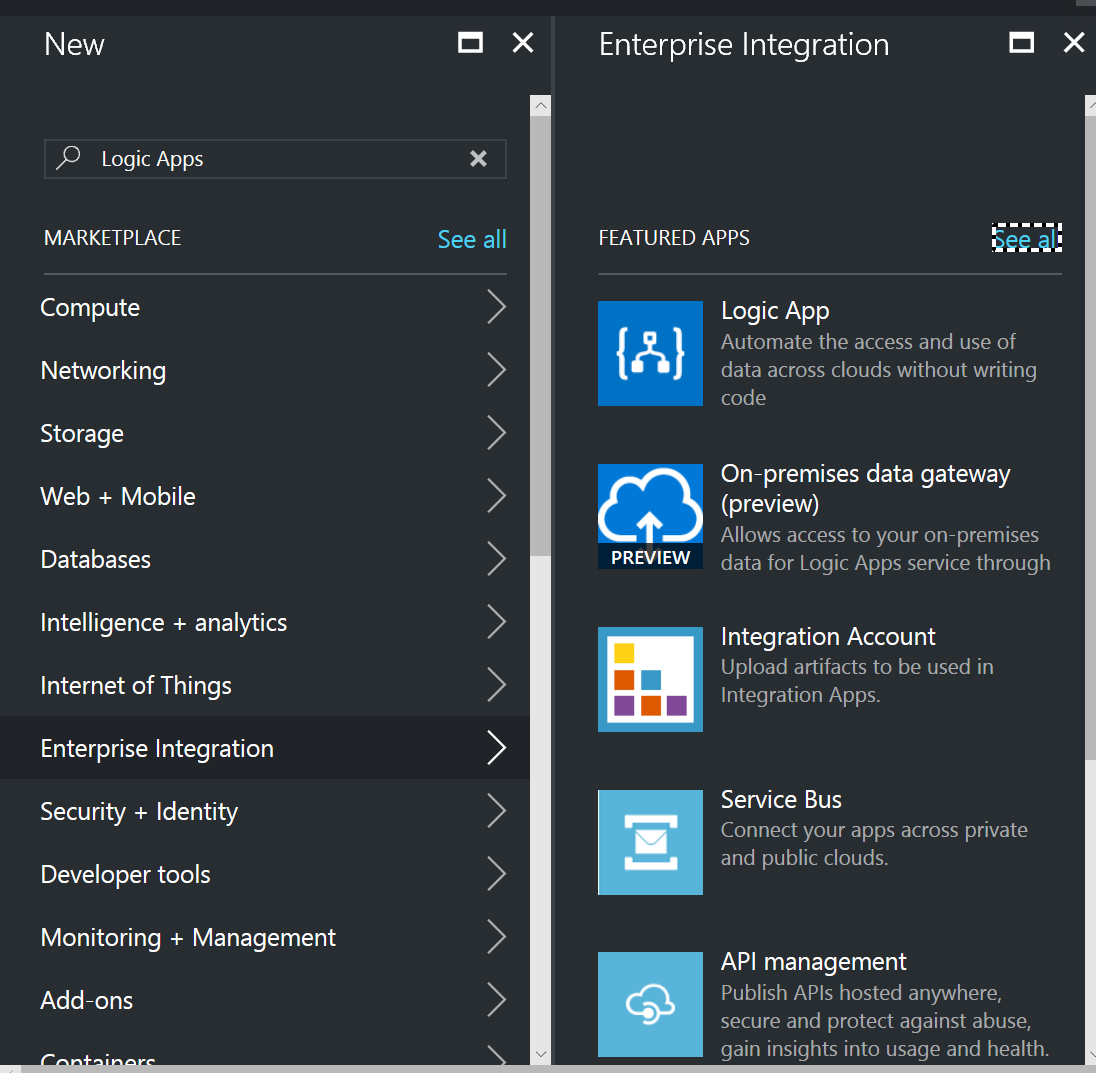


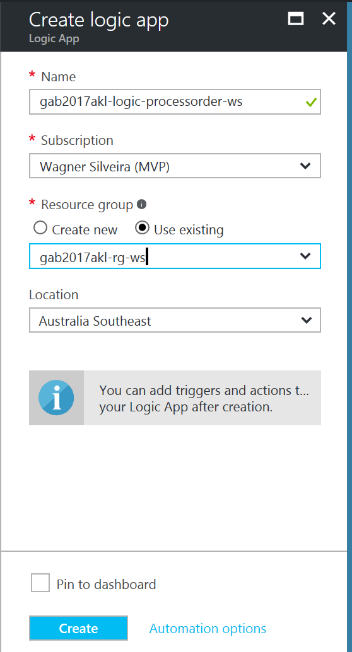
1. Change **Request body** to what is in the picture above.
2. Click **Run**.
3. Explore **Logs** and the **Output**.

## Provision a Logic App

The following steps describe how to provision a Logic App.

1. In the new Azure Portal click the **+**, Select Enterprise Integration then Logic Apps



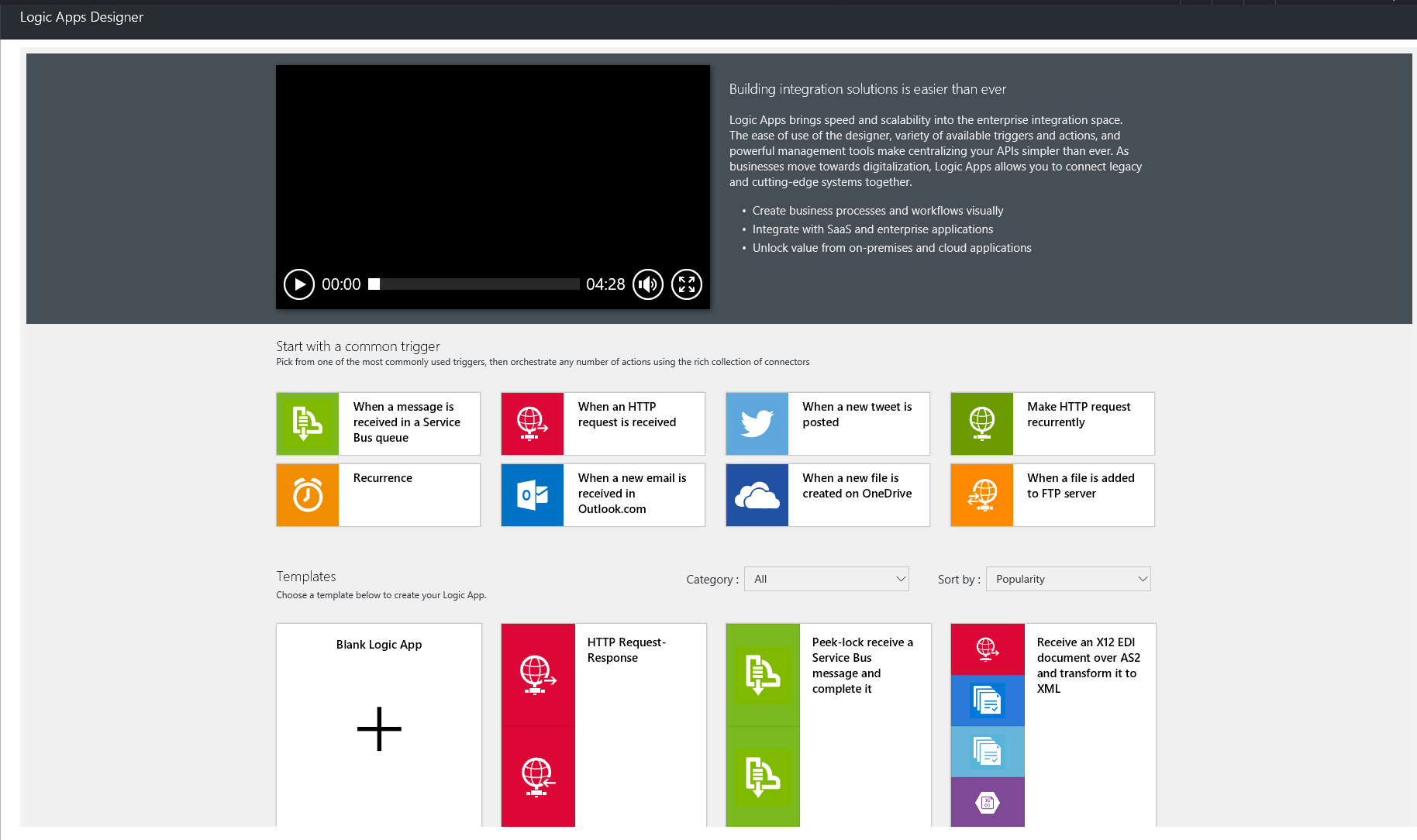
1. Within the Create Logic Apps blade, provide the following information:
   1. Name: **gab2017akl-logic-processorder-<ini>**
   2. Select the subscription used for this lab
   3. Select Use Existing and select **gab2017akl-rg-<ini>**
   4. The Location will be automatically selected from the resource group.
   5. Click in Create

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. A newly provisioned **Logic App** will automatically open in the template section, which display a video presenting a tutorial for new users, and a set of pre-defined templates. For this exercise, choose **Blank Logic App** and you will have access the **Logic Apps designer**.



1. Once in the Logic App designer surface, your first step is to add a trigger. Logic apps have a large selection of triggers from protocols like **HTTP**, **recurrence**, **WebHook**, to SaaS applications like Dynamics CRM (see [Workflow Actions and Triggers](https://msdn.microsoft.com/library/azure/mt643939.aspx)). In this lab, we will use the **HTTP** **trigger** (Request). First Select Request/Response in the services list, then click in the Request/Reponse Trigger

|  |  |
| --- | --- |
|  |  |

1. Once selected, to make the Logic App understand the messages being received as a trigger input and support the design time experience, the trigger should be configured with a JSON IETF JSON Schema Internet Draft Version 4. Logic apps can auto generate a schema for you based on a sample message. This can be achieved by clicking in the **Use sample payload to generate schema link** then provide the sample message below in the dialog box, clicking Done when completed.

|  |  |
| --- | --- |
|  |  |

**{**

"Order"**:{**

"Customer"**:{**

"Company"**:**"Contoso"**,**

"Email"**:**"John.Doe@hotmail.com"**,**

"CustomerNumber"**:**"JD-Contoso-001"**,**

"Address"**:{**

"Street"**:**"1 Shortland Street"**,**

"City"**:**"Auckland"**,**

"PostalCode"**:**"1000"**,**

"Country"**:**"New Zeland"

**}**

**},**

"Products"**:{**

"Product"**:[**

**{**

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

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

"Price"**:**123.45

**},**

**{**

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

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

"Price"**:**456.78

**}**

**]**

**},**

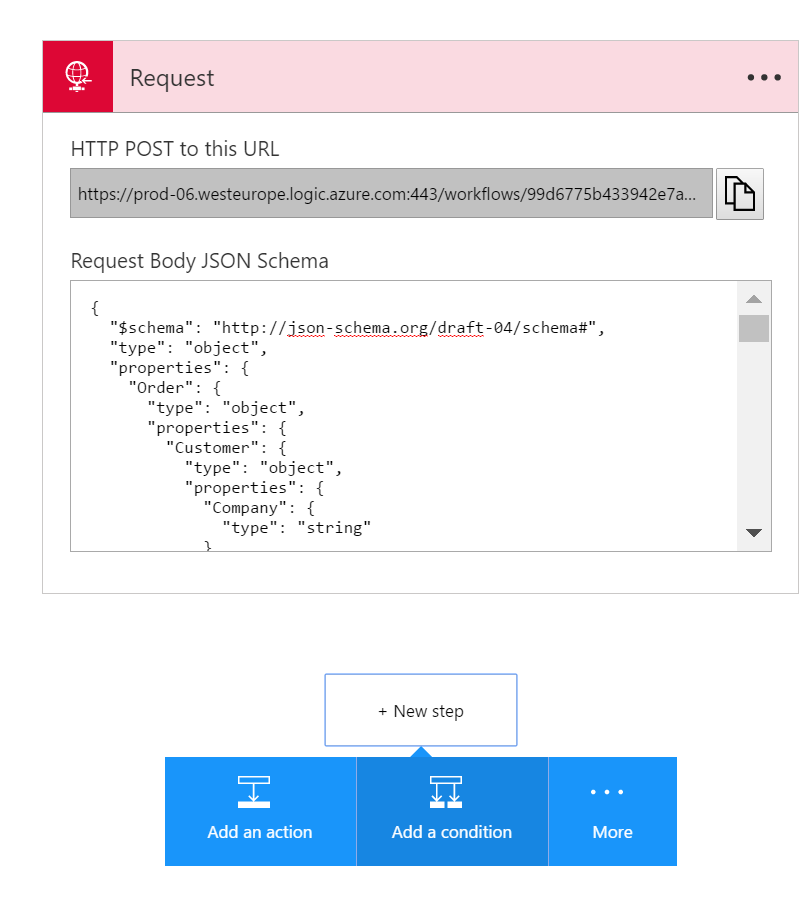
"OrderedDateTime"**:**"2017-04-22T13:30:00"**,**

"TotalInvoiceAmount"**:**2407.35

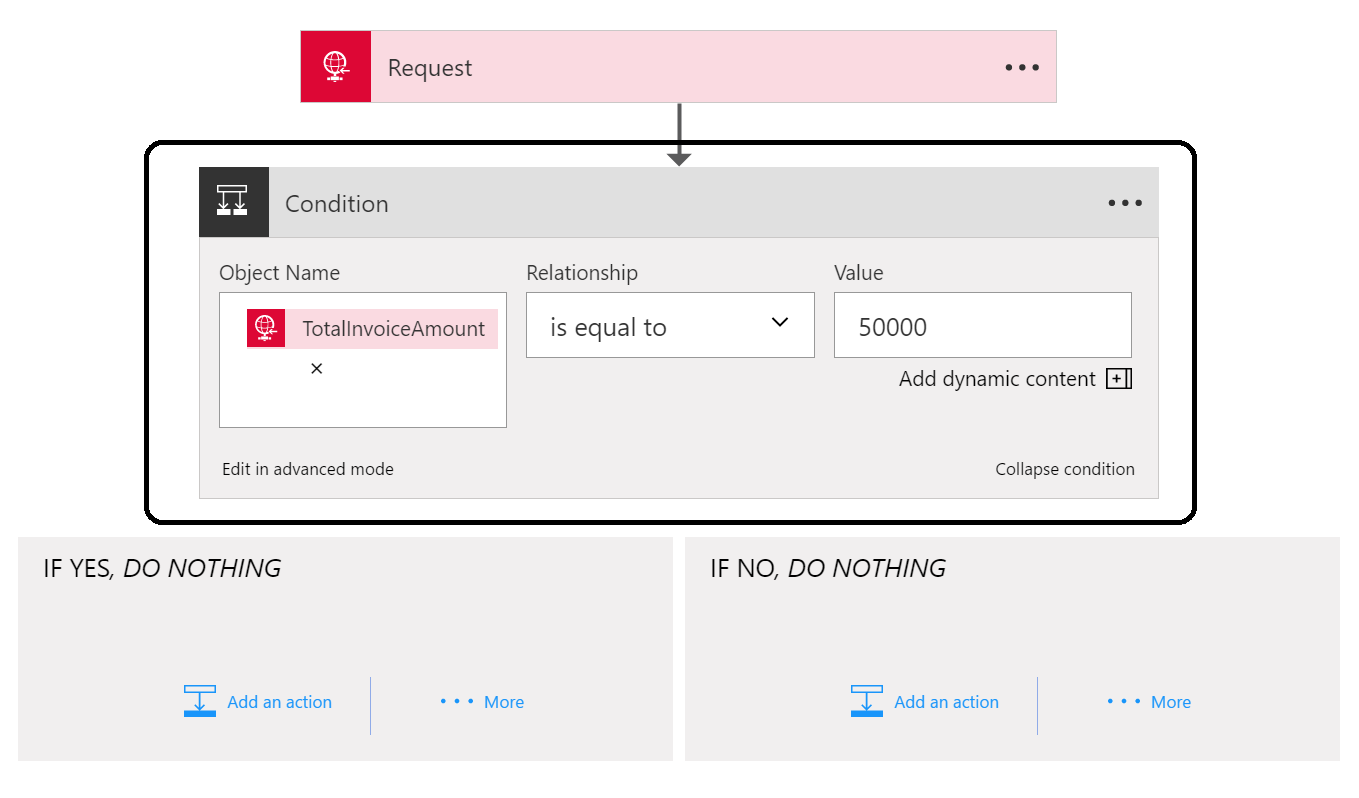
**}**

**}**

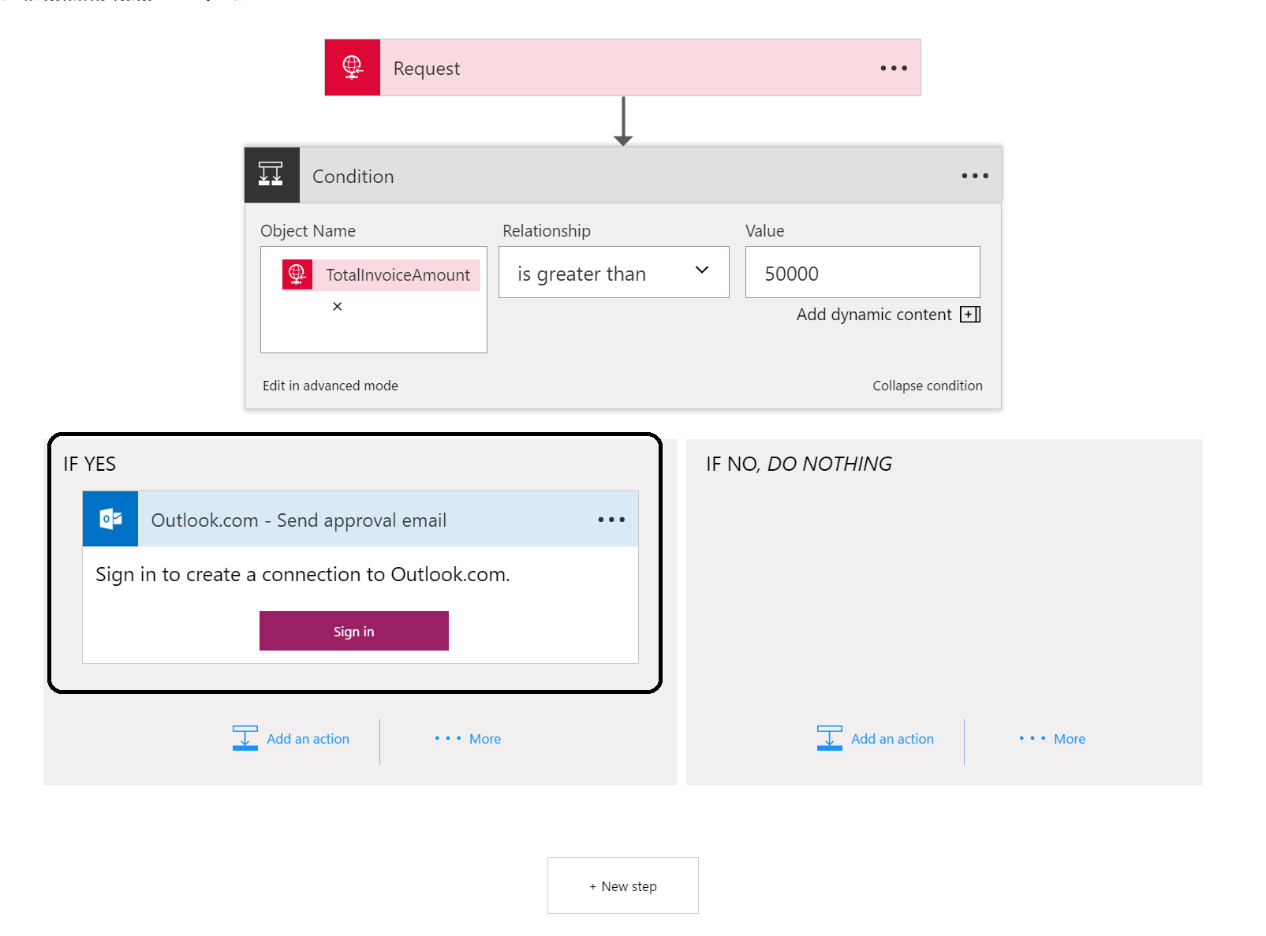
1. The next step is to add a **condition**.

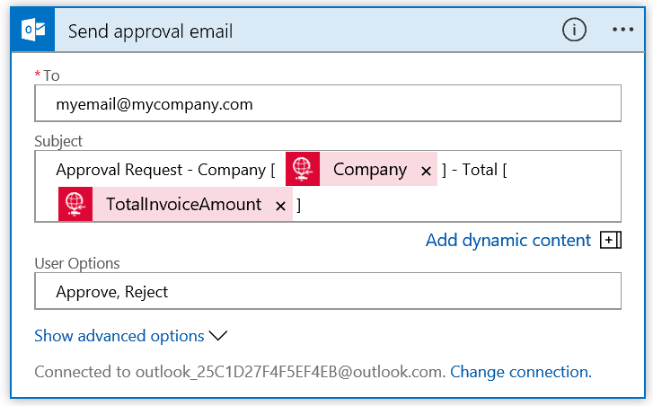


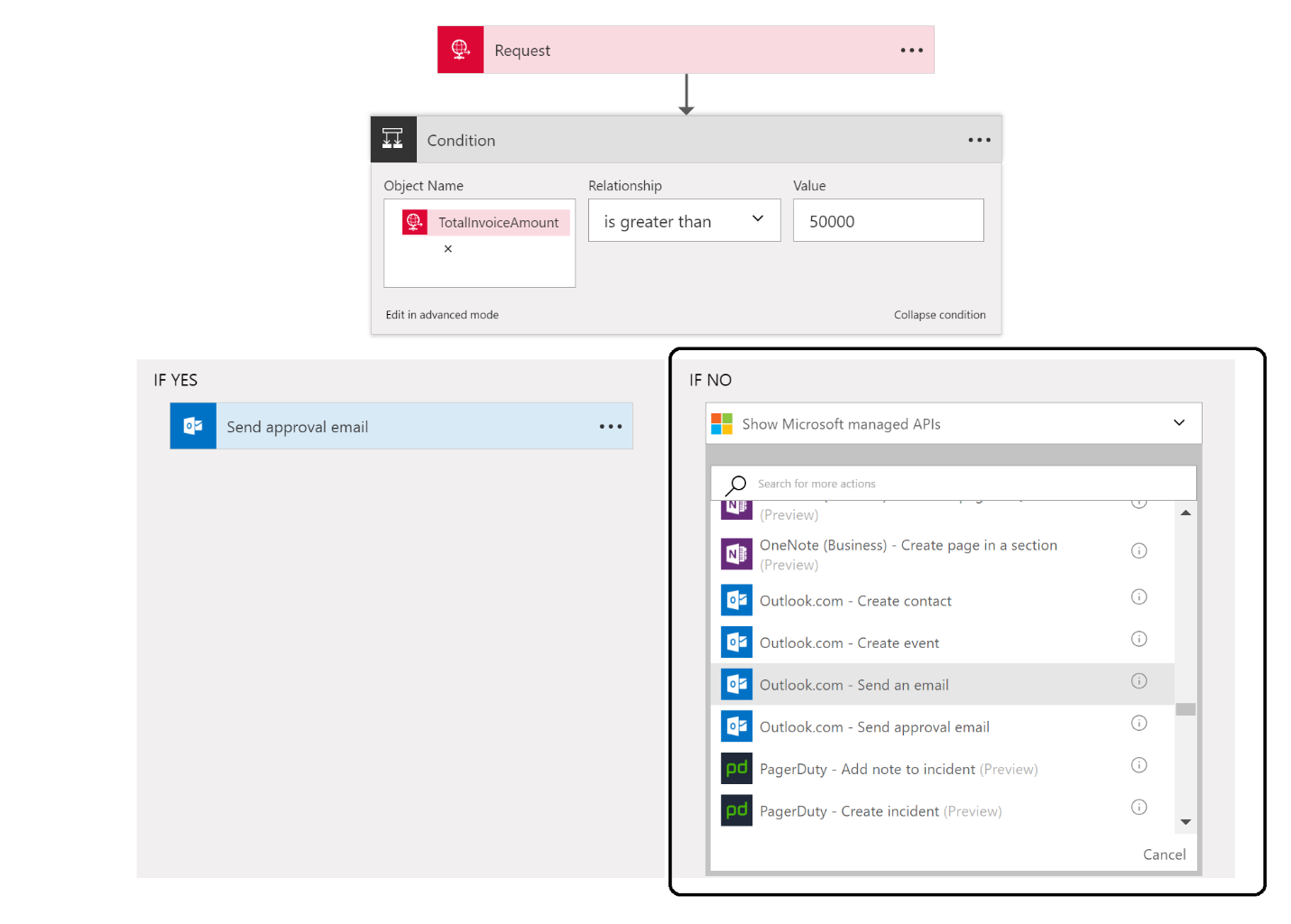
1. The **condition** will be to verify if the amount is higher than **50000**. You can use the Add dynamic content to find the elements from previous actions/triggers (e.g. TotalInvoiceAmount):



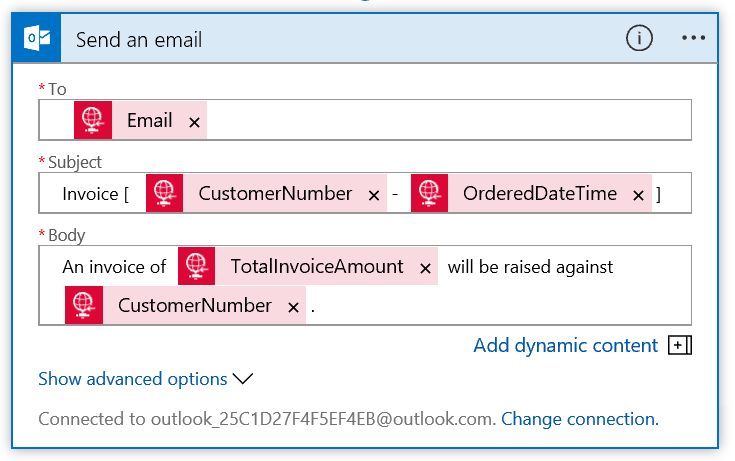
1. Add an **Outlook** (Send Approval Mail) **action** to the left branch.



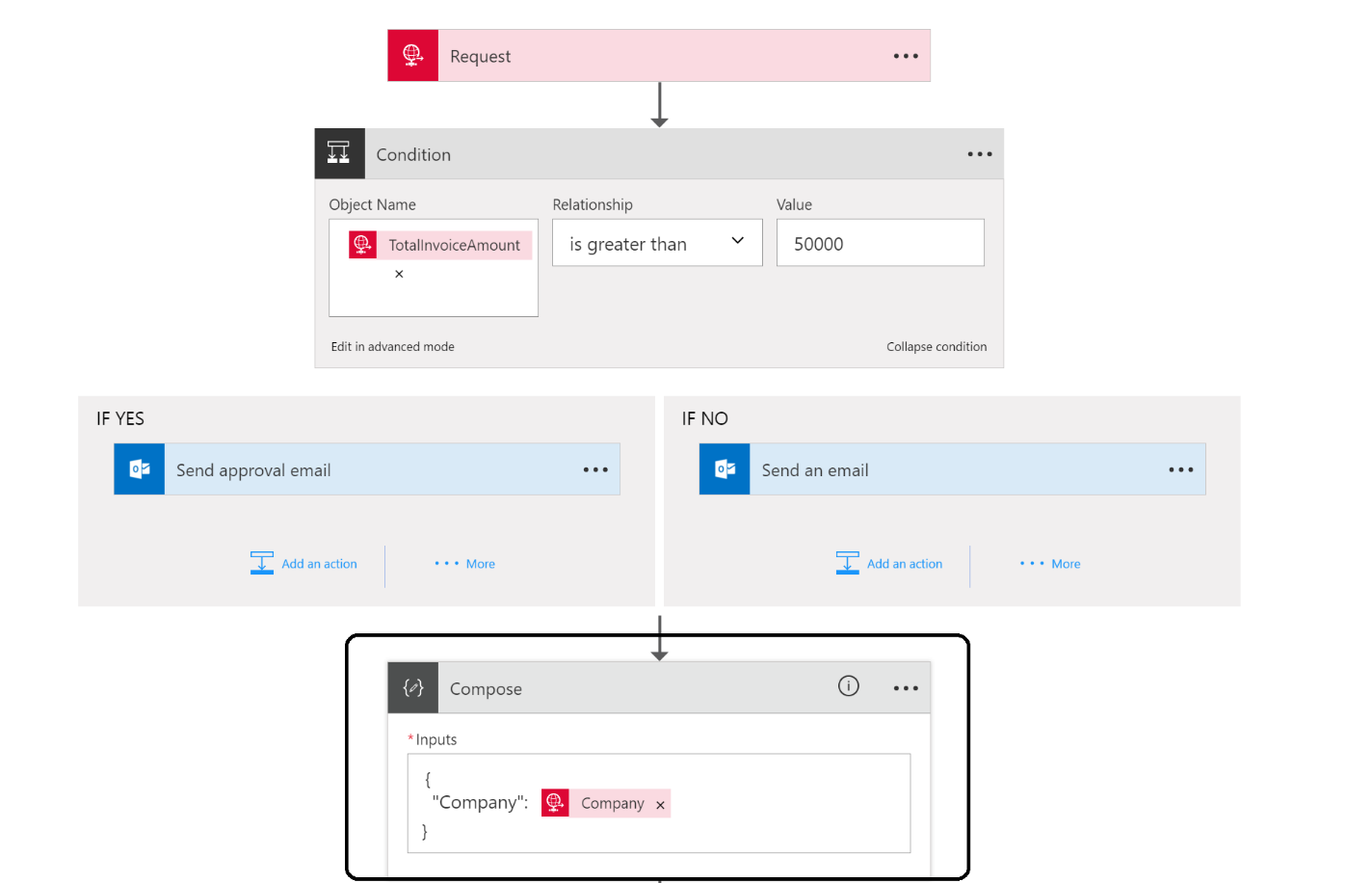
1. **Login** to your **Outlook.com** account. If you do not have one you can create one easily.
2. After the connection to your outlook has been established you can specify the details of the Send Approval Email action, providing the following information
   1. Your email address in the To Field,
   2. A subject line usint the tokens provided by the Add dynamic content
3. 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. Fill in the action by specifying the details as showed in the picture:



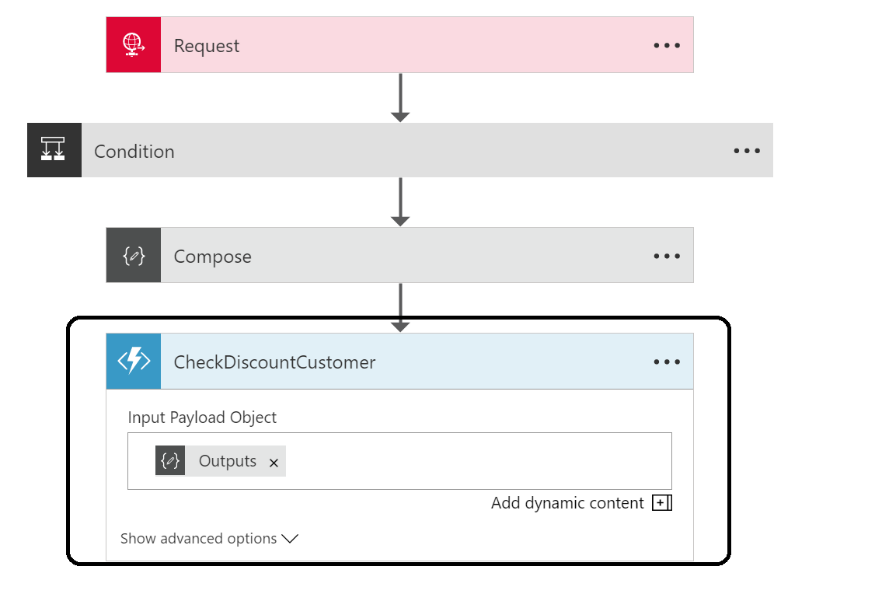
1. Add another action below the condition branches and search for a Compose action:



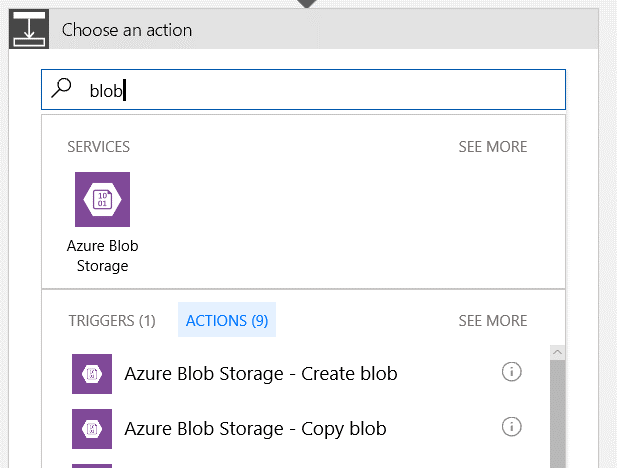
1. In the **Compose** Window create the input as above, making sure to wrap the Company field in quotation marks, otherwise the designer will report an error.
2. Add an action below the **Compose**. Select Azure Function, then Choose an Azure function. In the list of functions provided, select **gab2017akl-func-<ini>**, then **CheckDiscountCustomer** Function.

|  |  |
| --- | --- |
|  |  |
|  |  |

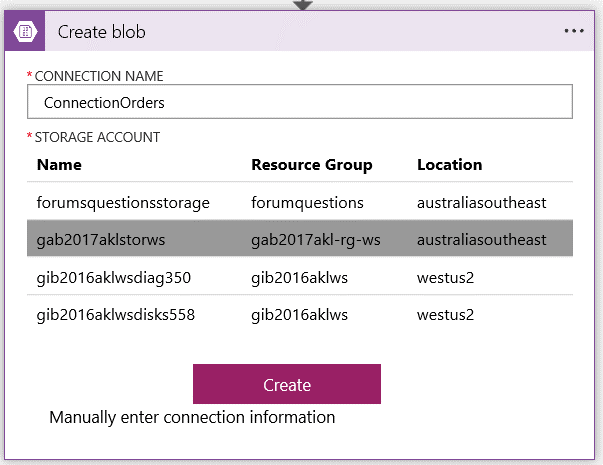
1. Select your custom created function.
2. Add the **Output** from the **Compose** in the **Input Payload Object,** from the dynamic content pane.



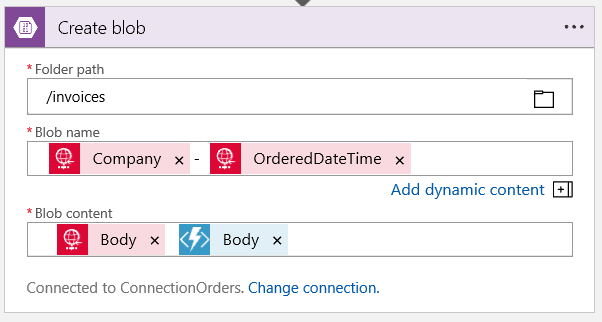
1. Search for blog, and select Create blob:



1. Setup the Blob Storage connection by giving a connection name, selecting the storage account created for this exercise – **gab2017aklstor<ini>**, and clicking in Create



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

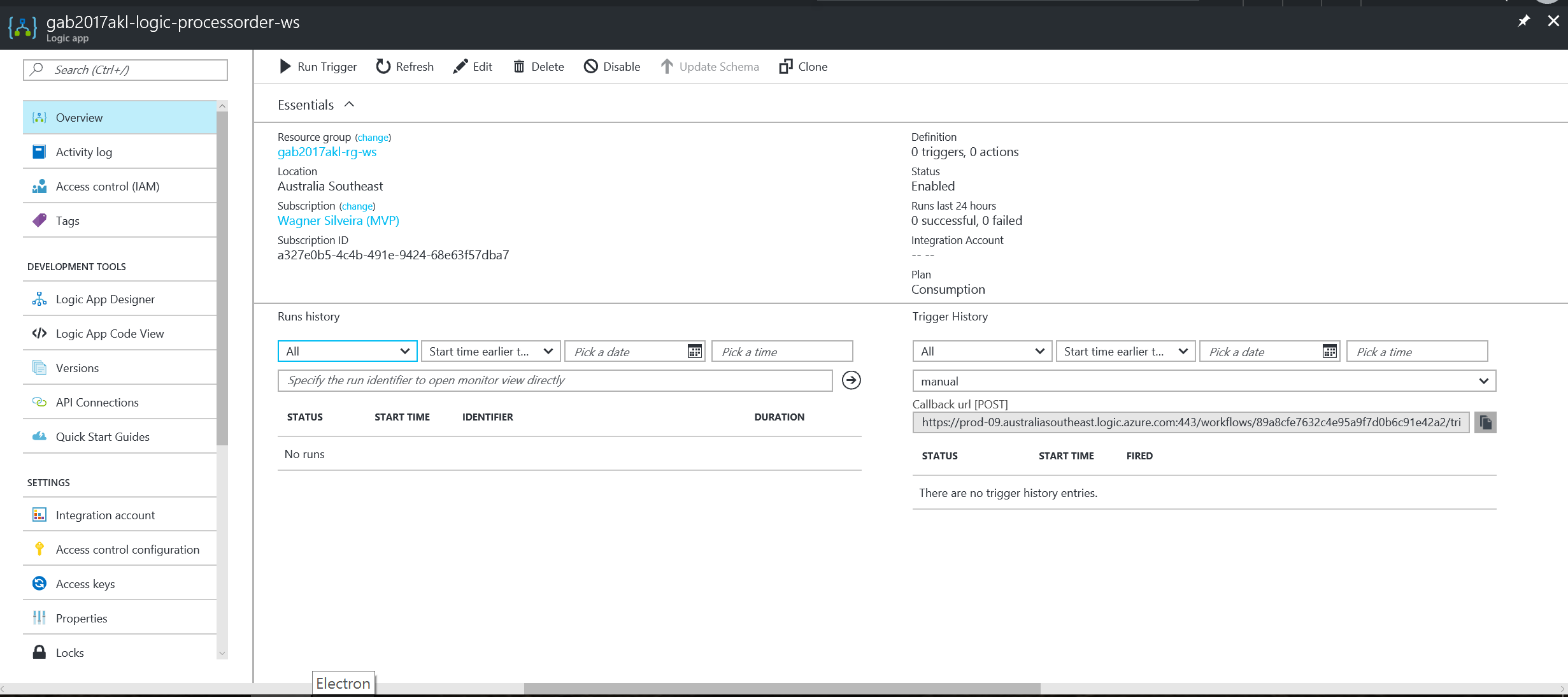


1. **Save** the Logic App Definition and close the designer.

## 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 access to the logic app trigger end point. You can find it in the trigger area. Copy it to clipboard by clicking in the button beside the URL



1. Open **postman**.
2. Copy address in address bar, select **POST** Method
3. For the body choose **raw** and content type **application/json**.
4. 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"**:**"Contoso"**,**

"Email"**:**"<your email here>"**,**

"CustomerNumber"**:**"JD-Contoso-001"**,**

"Address"**:{**

"Street"**:**"1 Shortland Street"**,**

"City"**:**"Auckland"**,**

"PostalCode"**:**"1000"**,**

"Country"**:**"New Zeland"

**}**

**},**

"Products"**:{**

"Product"**:[**

**{**

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

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

"Price"**:**123.45

**},**

**{**

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

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

"Price"**:**456.78

**}**

**]**

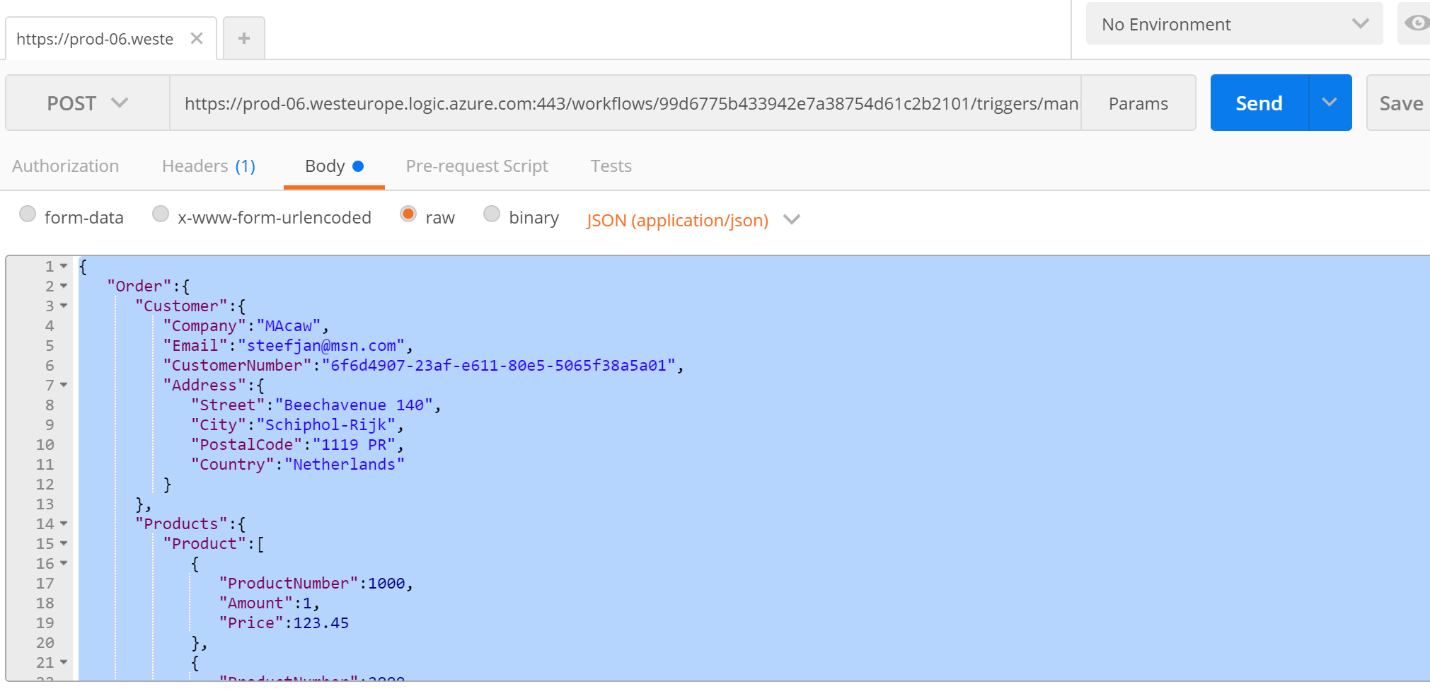
**},**

"OrderedDateTime"**:**"2017-04-22T13:30:00"**,**

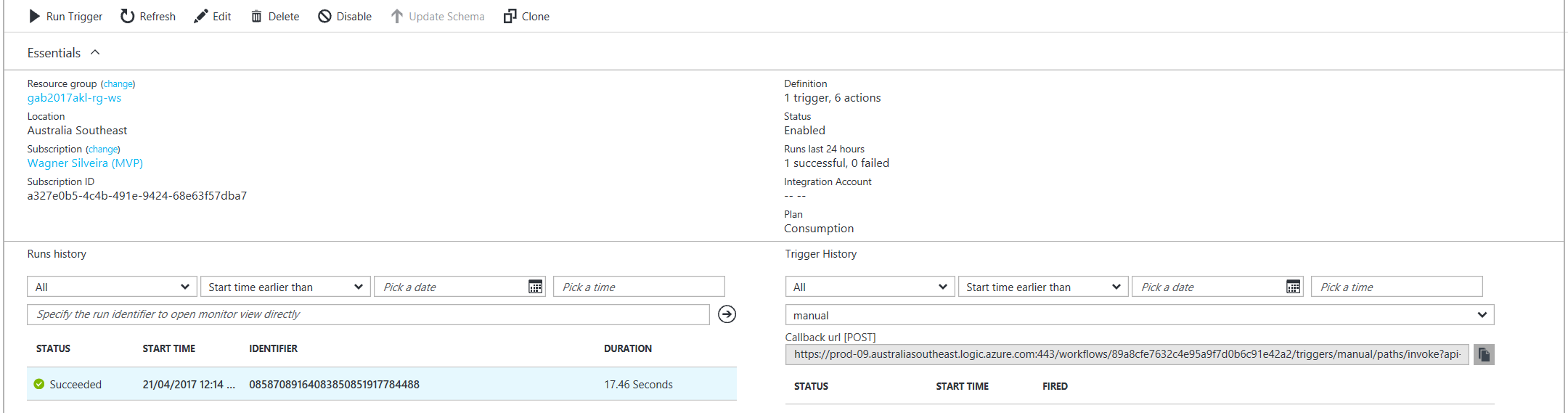
"TotalInvoiceAmount"**:**2407.35

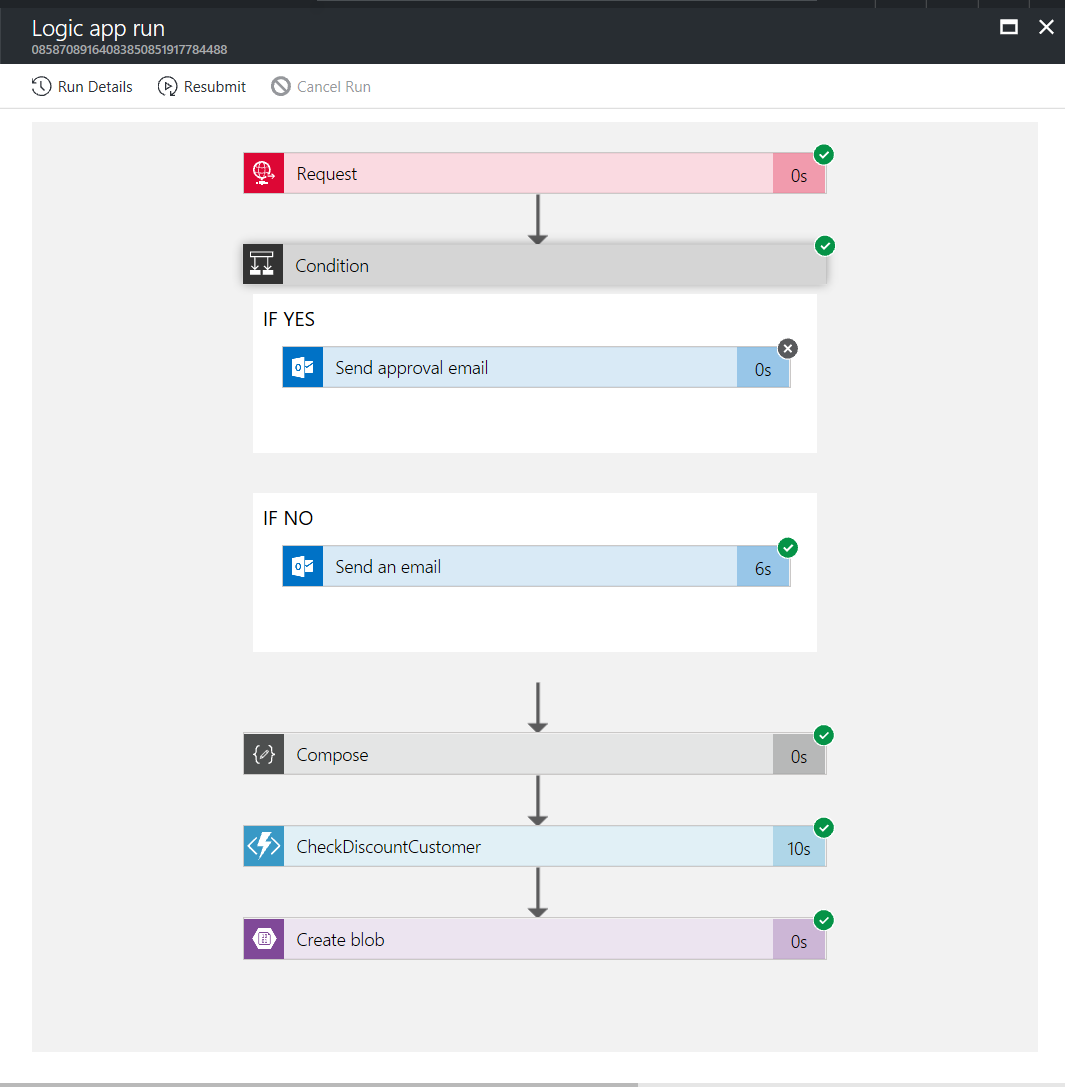
**}**

**}**

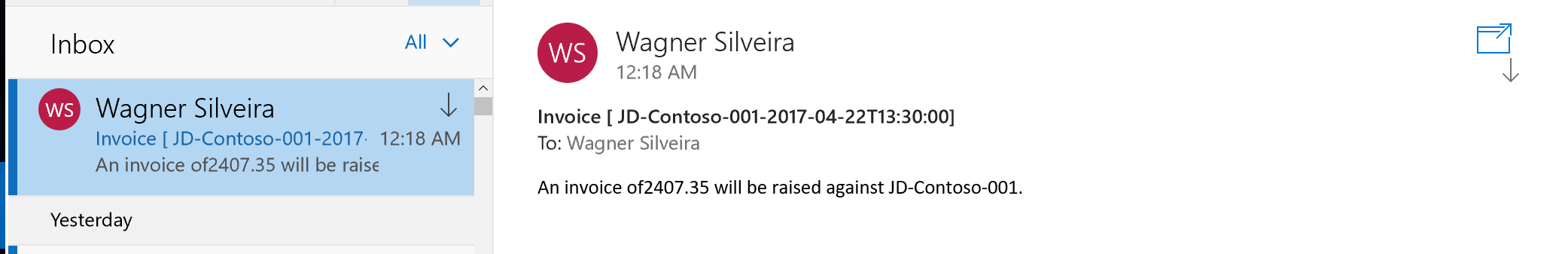
****

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.

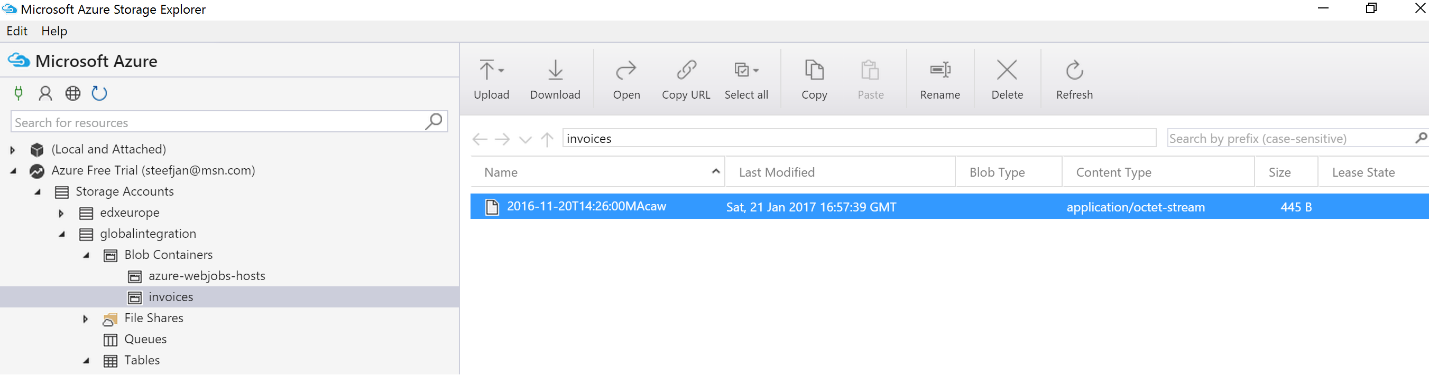




1. Look into your **mail box** and see if you have an **email**.



1. Check using the **Azure Storage 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.

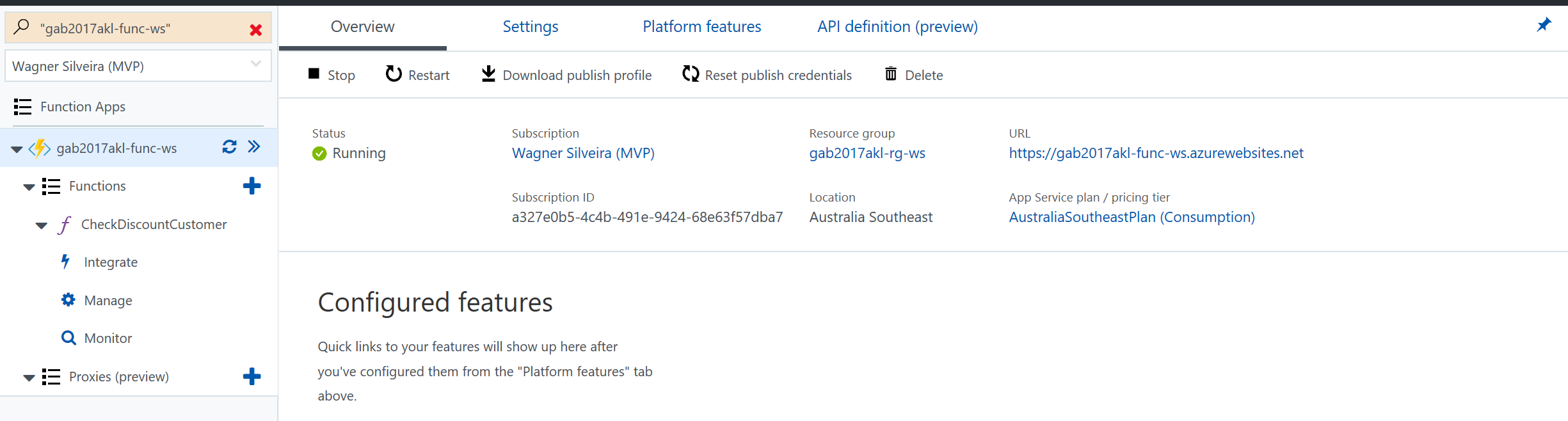
## Creating a Proxy Function

Proxy function allows you to specify endpoints on your function app that are implemented by another resource. You can use these proxies to break a large API into multiple function apps (as in a microservice architecture), while still presenting a single API surface for clients. You can also route calls to external resources that are a not an Azure Function, which will be demonstrated in this exercise. To make it easier to consume the Logic Apps, we will create a proxy endpoint that routes calls to that Logic Apps.

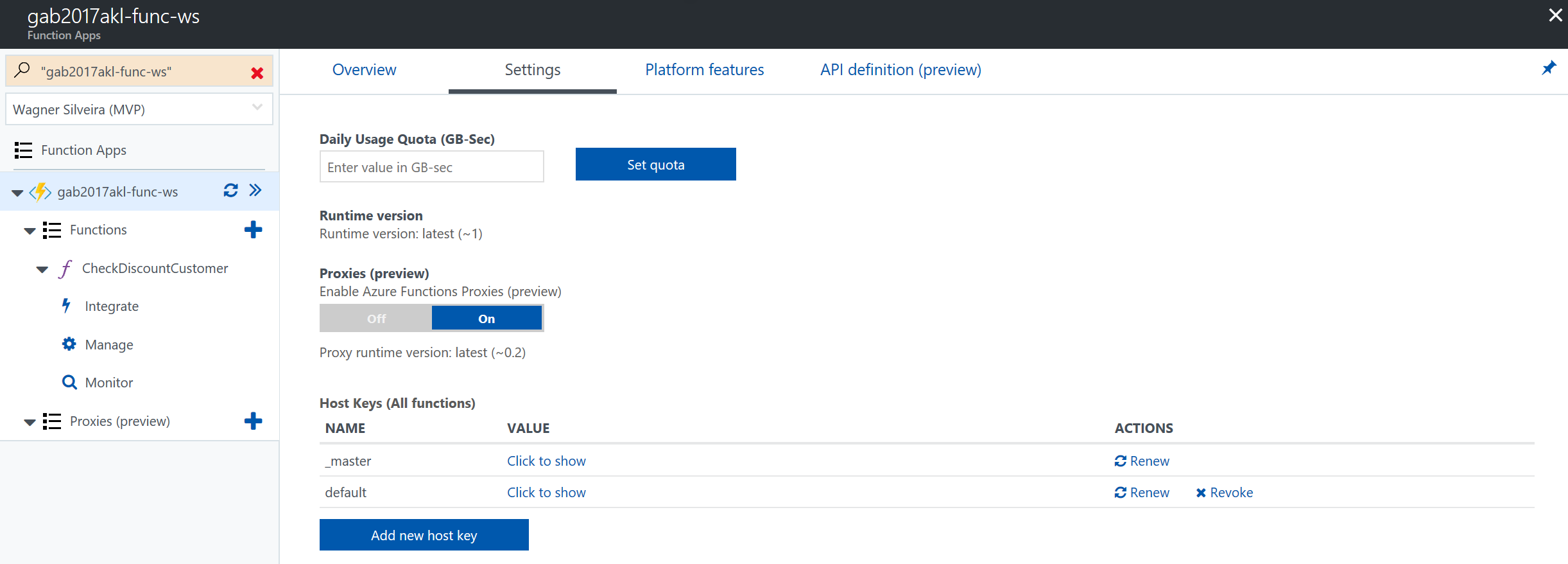
### Enabling Proxy Functionality

Before proxies are able to be used, this functionality needs to be enabled. To do this follow the instructions below.

1. Navigate to the Azure Function **gab2017akl-func-<ini>** and click in settings.



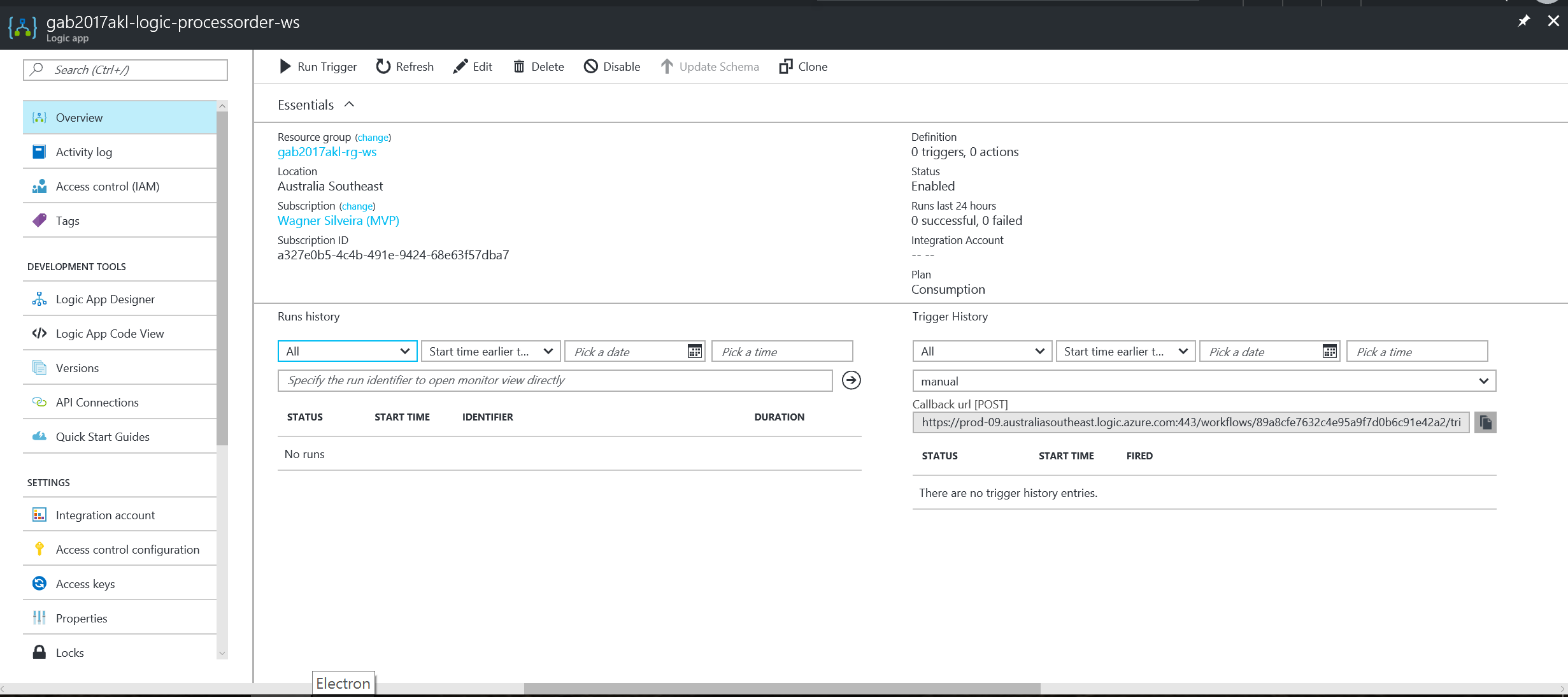
1. In the settings environment turn on the Proxy functionality, by clicking the **On** option in the Proxyies (preview) area.



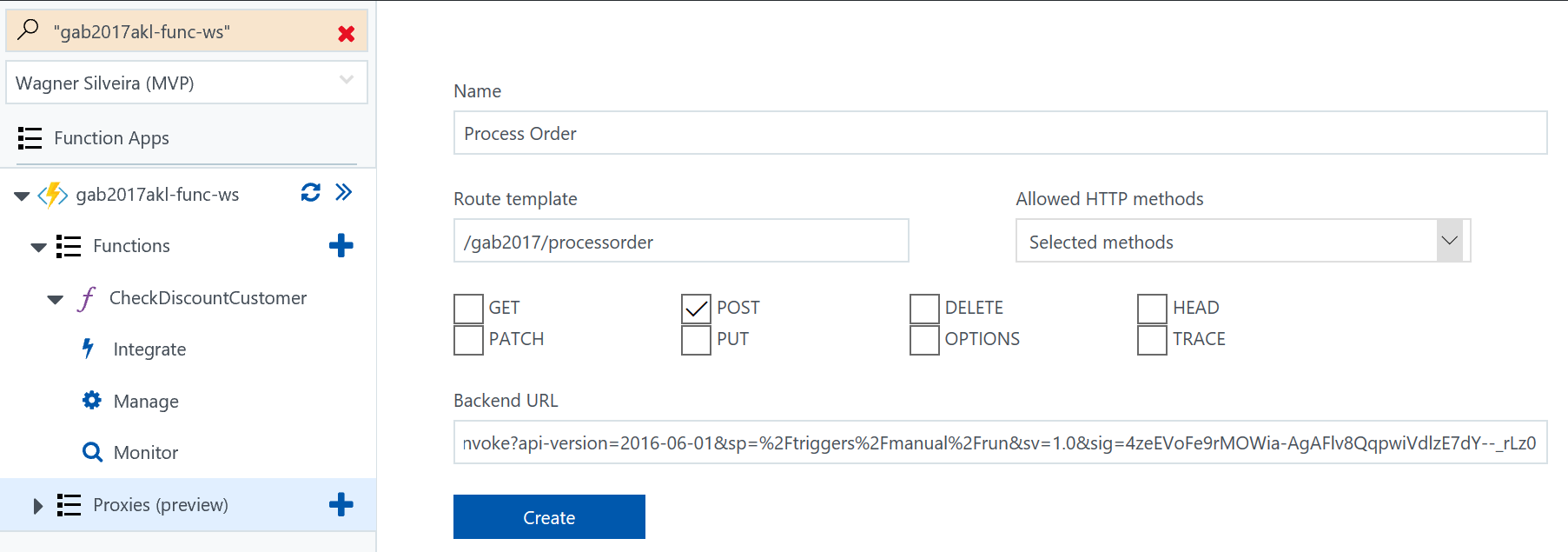
### Creating a Proxy Function

Since we are creating a proxy function, we first need to capture the URL of the endpoint this proxy will be redirected to. To do this.

1. Navigate to the logic apps **gab2017akl-logic-processorder-<ini>**
2. Click on the copy button of the logic app trigger to copy the URL

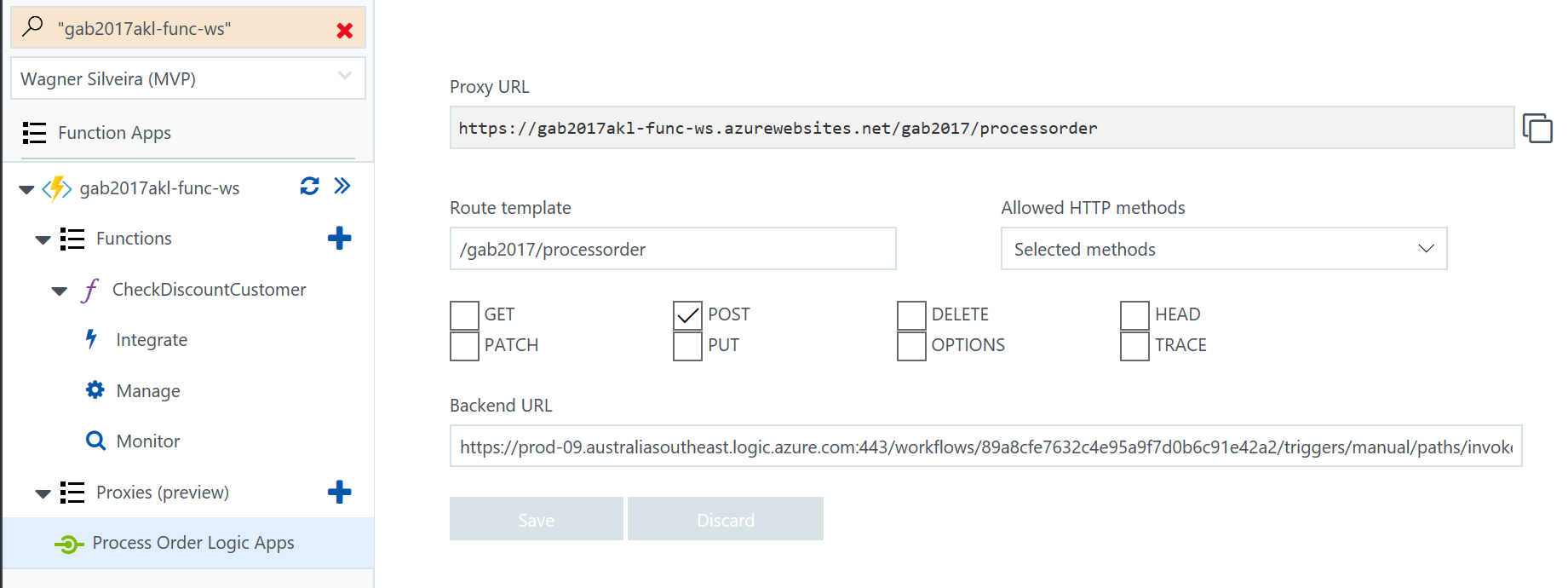


1. Navigate to the Navigate to the Azure Function **gab2017akl-func-<ini>** and click in   
   **Proxies (Preview) +**
2. In the Name field, add **Process Order Logic Apps**
3. In the Route template field, add **/gab2017/processorder**
4. In the Allowed HTTP methods, choose **Selected methods** and click in **POST**
5. In the Back URL paste the Logic Apps trigger URL captured previously. Remove the port number from the URL (**:443**)[[1]](#footnote-1)
6. Click in **Create**



### Testing the Proxy Function

1. Copy the proxy function URL, by clicking in the copy button:



1. Open **postman**.
2. Copy address in address bar, select **POST** Method
3. For the body choose **raw** and content type **application/json**.
4. 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"**:**"Contoso"**,**

"Email"**:**"<your email here>"**,**

"CustomerNumber"**:**"JD-Contoso-001"**,**

"Address"**:{**

"Street"**:**"1 Shortland Street"**,**

"City"**:**"Auckland"**,**

"PostalCode"**:**"1000"**,**

"Country"**:**"New Zeland"

**}**

**},**

"Products"**:{**

"Product"**:[**

**{**

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

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

"Price"**:**123.45

**},**

**{**

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

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

"Price"**:**456.78

**}**

**]**

**},**

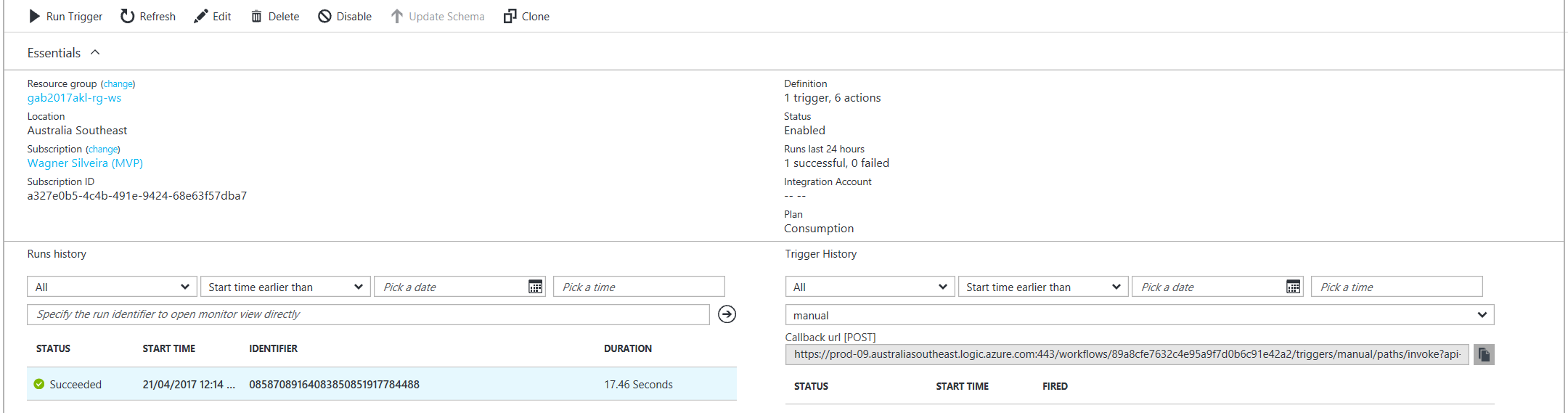
"OrderedDateTime"**:**"2017-04-22T13:30:00"**,**

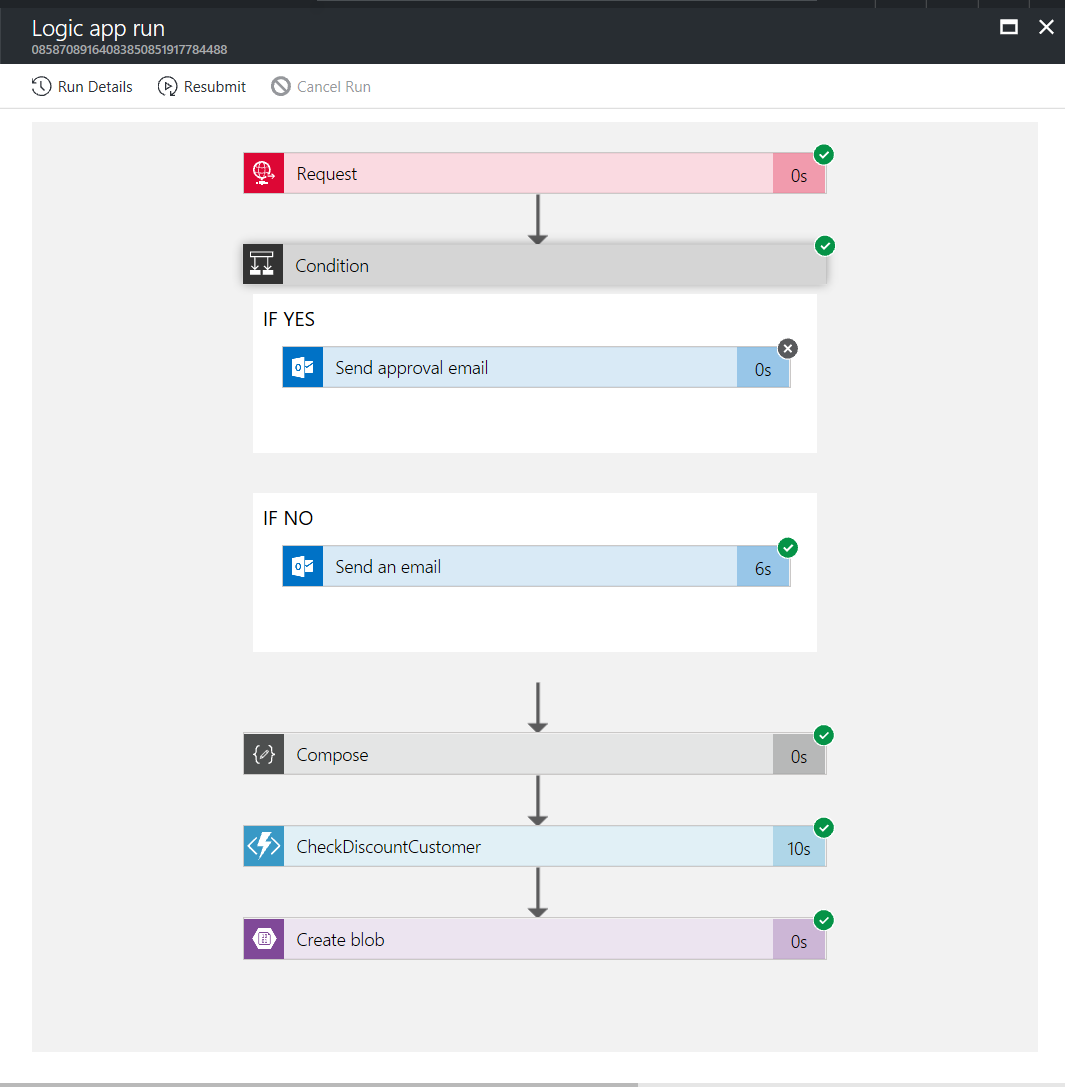
"TotalInvoiceAmount"**:**2407.35

**}**

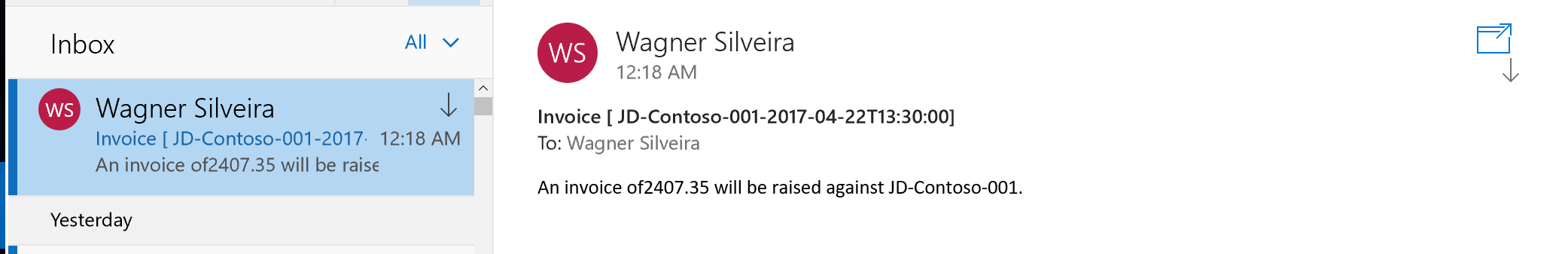
**}**

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. This step is a workaround for a bug currently in the preview functionality. [↑](#footnote-ref-1)