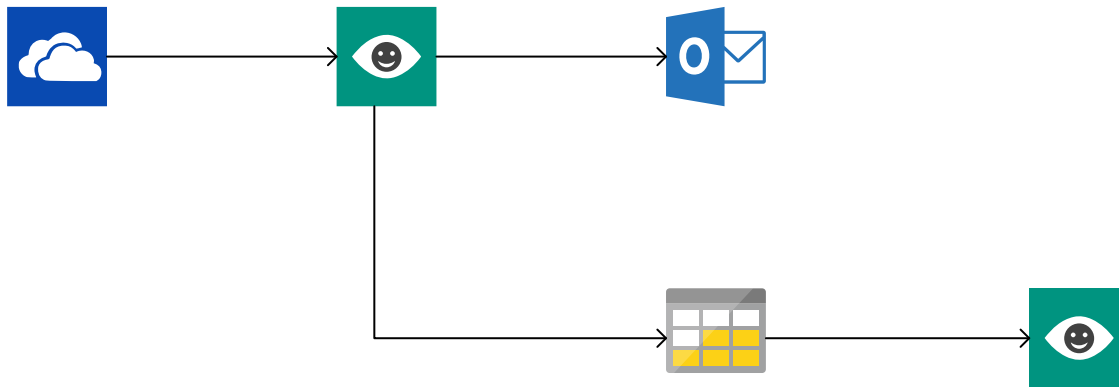


Lab- Recognizing people with Cognitive Services Face API

Author: Eldert Grootenboer

Objective

In this lab, we will build a solution for recognizing faces using the Face API. A picture will be uploaded to OneDrive, from where it will be picked up by a Logic App. Here we will call the Face API, and determine if this is a known person. If it is not a known person, we will send it to Azure Table Storage where we can assign a person.



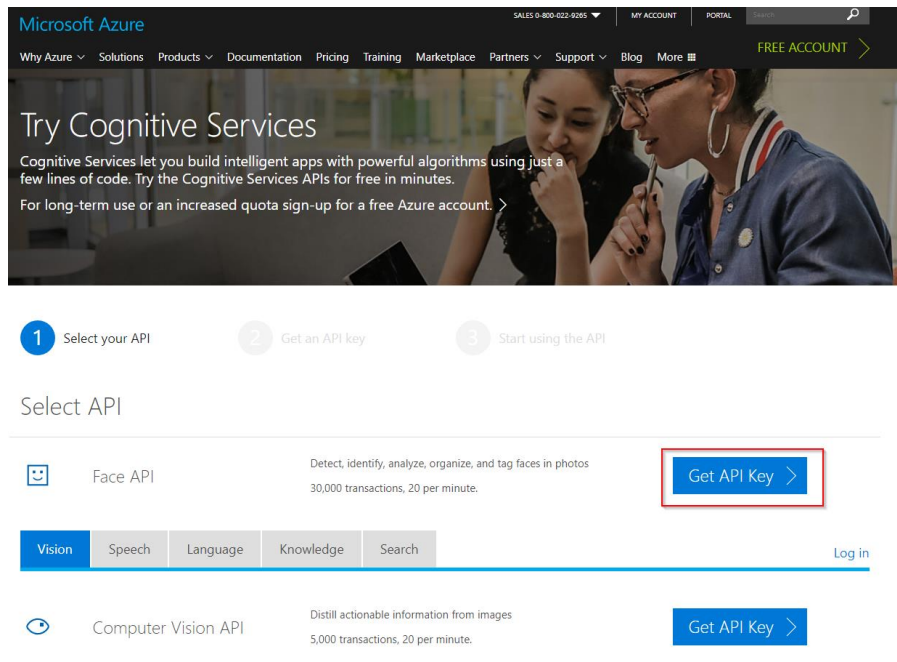
Prerequisites

- Azure Subscription
- Azure Storage Explorer: <https://azure.microsoft.com/en-us/features/storage-explorer/>

GIBC GLOBAL INTEGRATION BOOTCAMP

Create Face API Account

Start by creating a Face API account. Navigate to <https://azure.microsoft.com/en-us/try/cognitive-services/?api=face-api> and select Get API Key.



Accept the conditions and login using your preferred method.

Microsoft Cognitive Services Terms

Please review the service terms for your free trial.

☒ I agree to the Microsoft [Cognitive Services Terms](#) and [Microsoft Privacy Statement](#). Please note that these are not the typical terms for Azure services. The Microsoft Trust Center does not apply.

Select your Country / Region

Netherlands

Microsoft may use your contact information to provide updates and special offers about Artificial Intelligence and other Microsoft products and services. You can unsubscribe at any time. To learn more you can read the [privacy statement](#).

☒ I accept

Next >

Grab the endpoint and key, you will need this later on.

GIBC GLOBAL INTEGRATION BOOTCAMP



Successfully added Face API to your subscription.

Your APIs

Hello eldert@eldert.net ([Log out](#))



Face API

This API key is currently active
30 days remaining

Detect, identify, analyze, organize, and tag faces in photos
30,000 transactions, 20 per minute.

[Quick-start Guide >](#)

Endpoint: <https://westcentralus.api.cognitive.microsoft.com/face/v1.0>

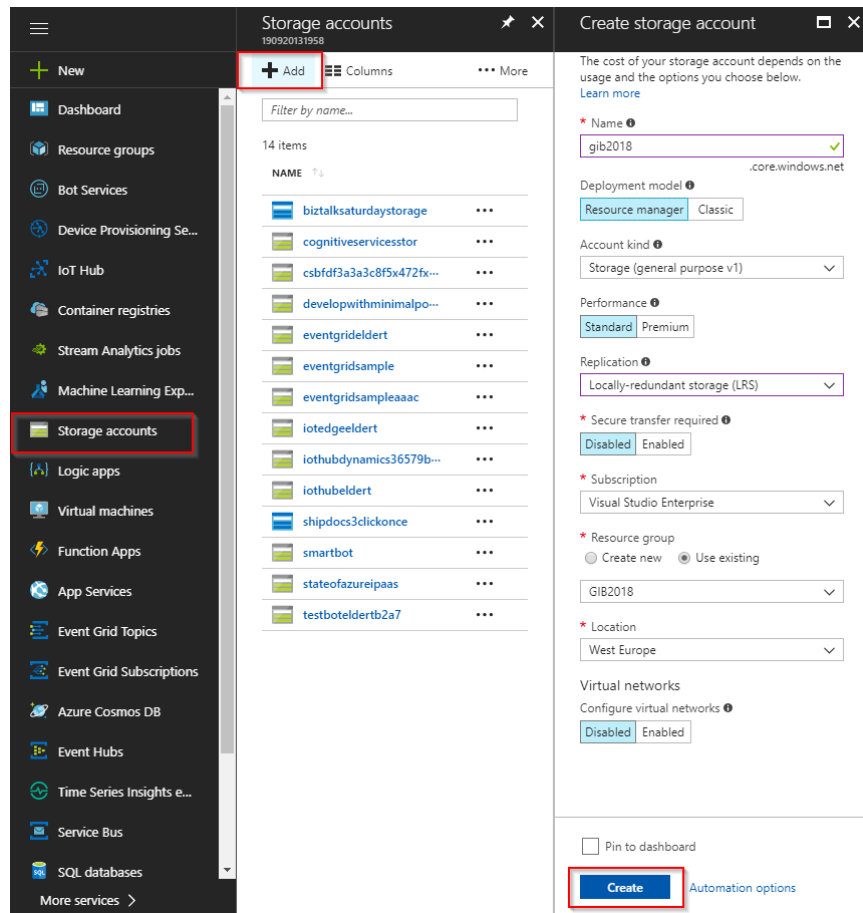
Key 1: [5601b0c0-2f0e-4b0e-8b0e-1a2b3c4d5057](#)

Key 2: [b5d1e0c0-2f0e-4b0e-8b0e-1a2b3c4d5050](#)

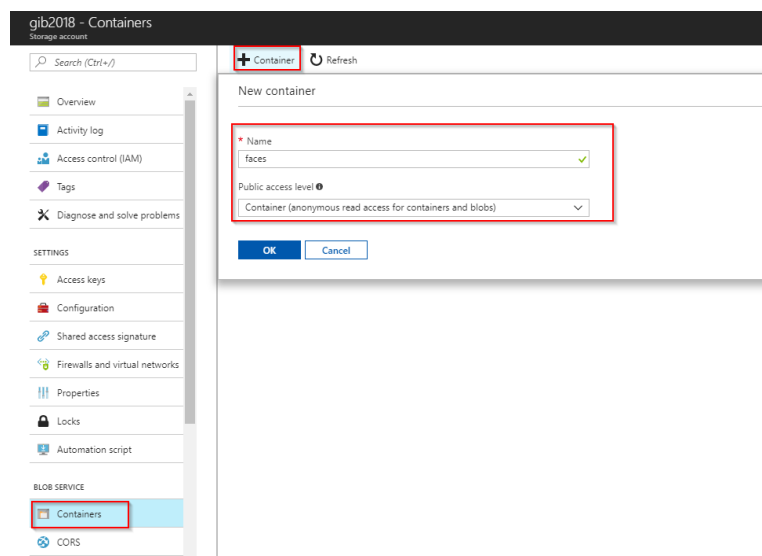
GIBC GLOBAL INTEGRATION BOOTCAMP

Create storage account

Now navigate to the [Azure portal](#), and create a new storage account.



Once created, open the new Storage Account, and create a new blob storage container. Make sure to allow anonymous access.



GIBC GLOBAL INTEGRATION BOOTCAMP

Grab the endpoint of your storage account, you will need this later on.

The screenshot shows the Azure portal interface for a storage account named 'gib2018'. At the top, there are buttons for '+ Container' and 'Refresh'. On the left, a sidebar lists account details: 'Storage account gib2018', 'Status: Primary: Available', 'Location: West Europe', 'Subscription (change): Visual Studio Enterprise', and 'Subscription ID: fdf3a3a3-c8f5-472f-8367-6a9a4a6c11a9'. A red box highlights the 'Blob service endpoint' as 'https://gib2018.blob.core.windows.net/'. Below this is a search bar 'Search containers by prefix' and a table with columns: NAME, LAST MODIFIED, PUBLIC ACCESS L..., and LEASE STATE. The table contains one entry: 'faces' with a last modified date of '4-2-2018 1:52:19 p.m.', public access of 'Container', and lease state of 'Available'.

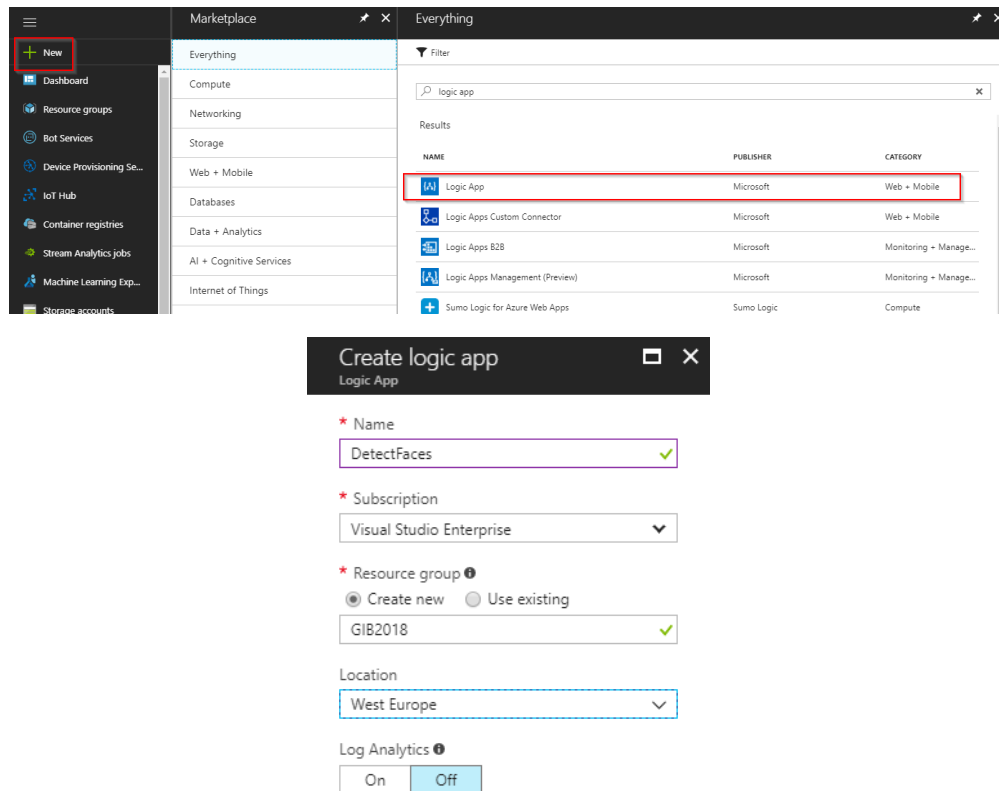
Now in the same storage account, add a Table storage.

The screenshot shows the 'gib2018 - Tables' interface in the Azure portal. On the left, a sidebar lists various services: 'Add Azure Search', 'Metrics', 'Usage', 'FILE SERVICE' (Files, CORS, Metrics), and 'TABLE SERVICE' (Tables, CORS, Metrics). The 'Tables' option under 'TABLE SERVICE' is highlighted with a red box. The main area shows the 'Add table' dialog with a 'Table name' field containing 'ReviewFaces' and a green checkmark. Below the dialog is a table with columns 'TABLE' and 'URL', which is currently empty with the message 'You don't have any tables yet.'.

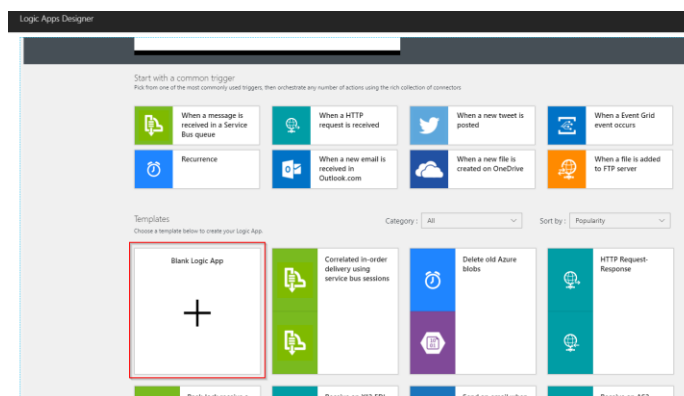
GIBC GLOBAL INTEGRATION BOOTCAMP

Create Logic App

Now in the portal, create a Logic App, in which we will set up the flow.

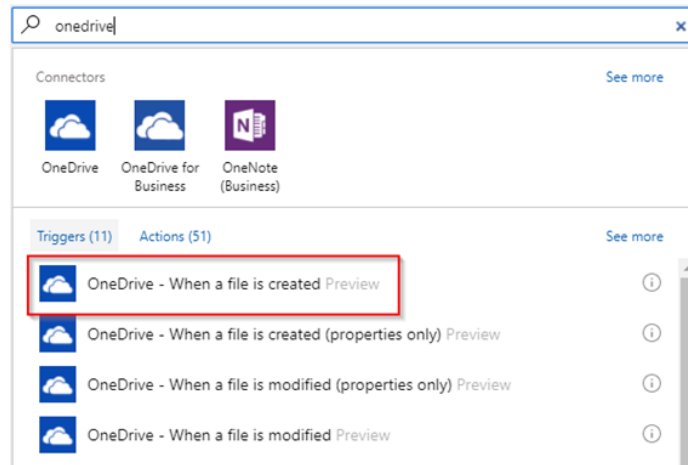


Create a blank logic app.

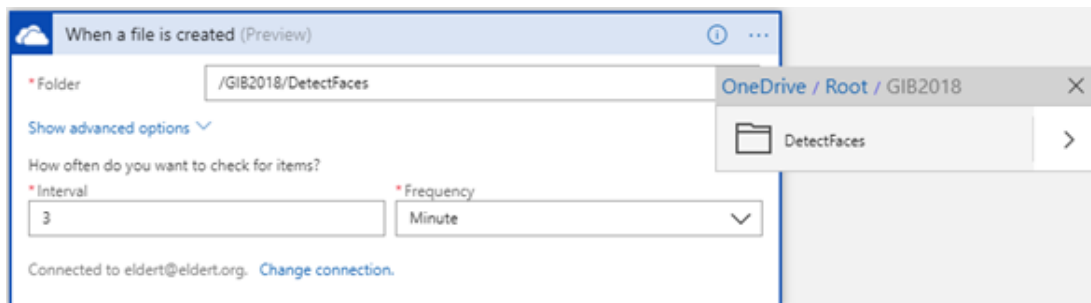


We will start with a OneDrive trigger, which will check for a file being created.

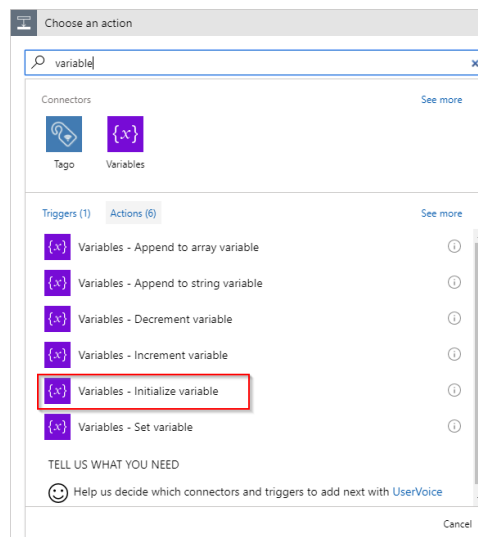
GIBC GLOBAL INTEGRATION BOOTCAMP



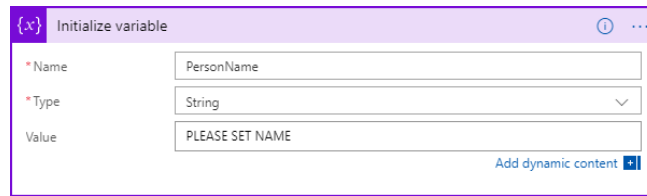
Sign in using your onedrive account, and select the folder from which you will want to pick up the images. Set the interval to 1 minute.



Next we will add a string variable which will be used to hold the name of the person. Initialize the variable with the value PLEASE SET NAME.



GIBC GLOBAL INTEGRATION BOOTCAMP



Initialize variable

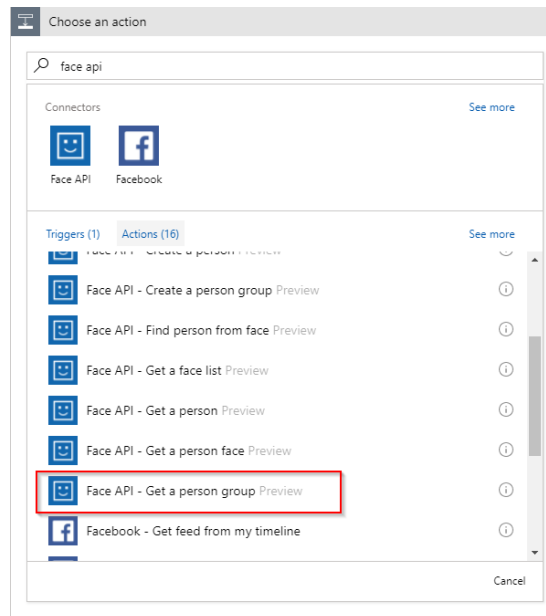
*Name: PersonName

*Type: String

Value: PLEASE SET NAME

[Add dynamic content](#)

Create a scope and name it Create person group if not exists. Inside this scope, add an action to get a person from the Face API.



Choose an action

face api

Connectors: Face API, Facebook

Triggers (1): Create person group if not exists

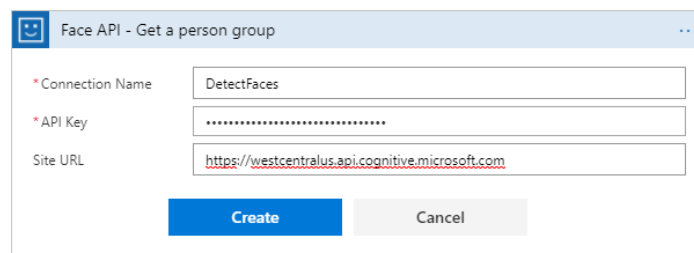
Actions (16):

- Face API - Create a person group
- Face API - Find person from face
- Face API - Get a face list
- Face API - Get a person
- Face API - Get a person face
- Face API - Get a person group (Preview) - **Selected**
- Facebook - Get feed from my timeline

[See more](#)

Cancel

Set up the connection using the site url and API key you got by creating your Face API account.



Face API - Get a person group

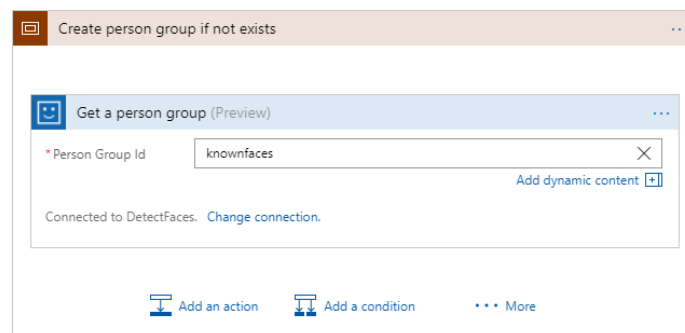
*Connection Name: DetectFaces

*API Key:

Site URL: <https://westcentralus.api.cognitive.microsoft.com>

[Create](#) [Cancel](#)

The person group will be called knownfaces.



Create person group if not exists

Get a person group (Preview)

*Person Group Id: knownfaces

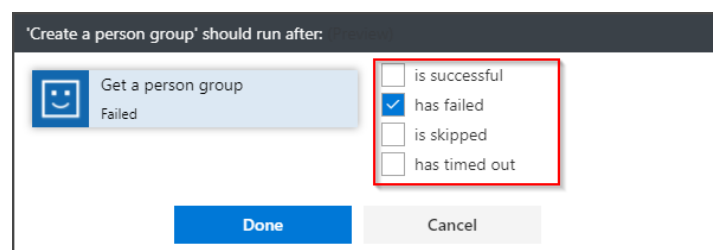
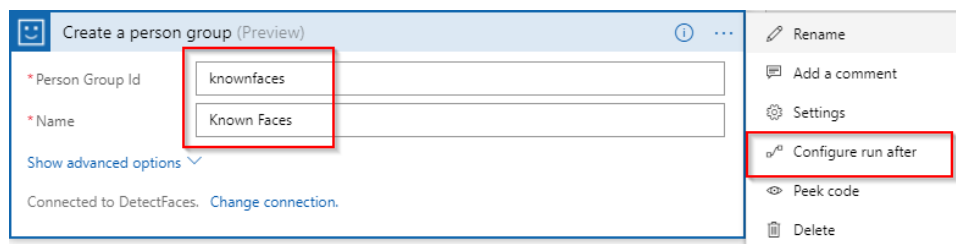
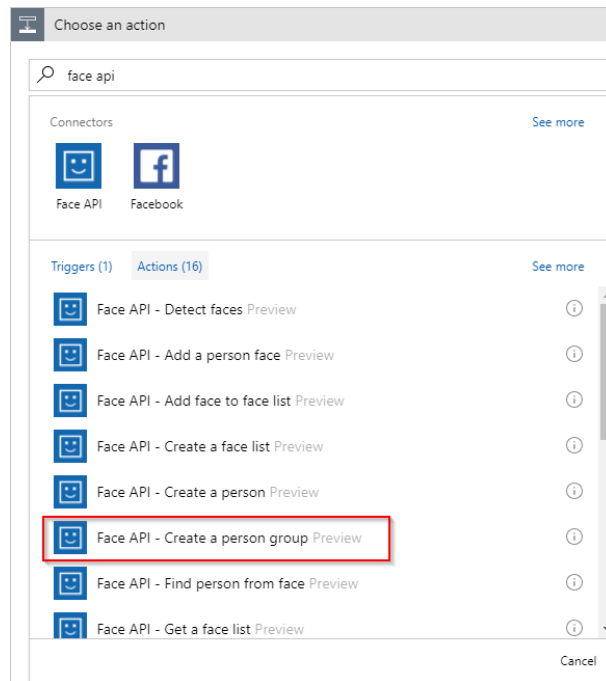
[Add dynamic content](#)

Connected to DetectFaces. [Change connection.](#)

[Add an action](#) [Add a condition](#) [More](#)

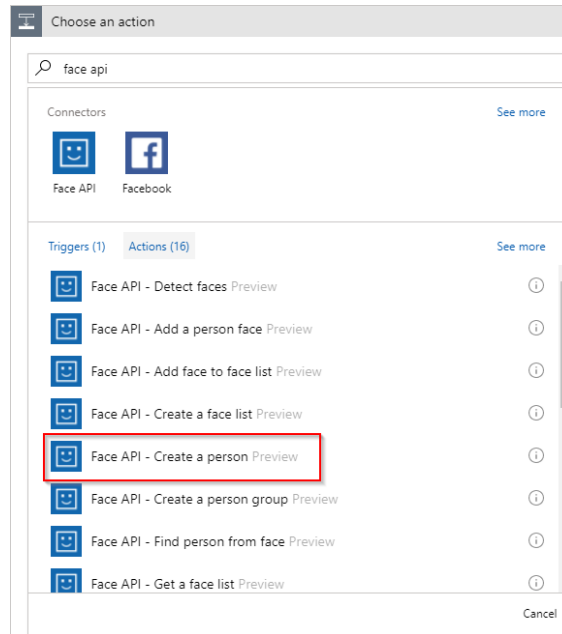
GIBC GLOBAL INTEGRATION BOOTCAMP

Now in case the person group does not exist, we will need to create it. For this, add a Create a person group action, and configure the run after for this action to has failed. This will make sure that if the person group is not found, throwing an error, this action is called to create the person group.

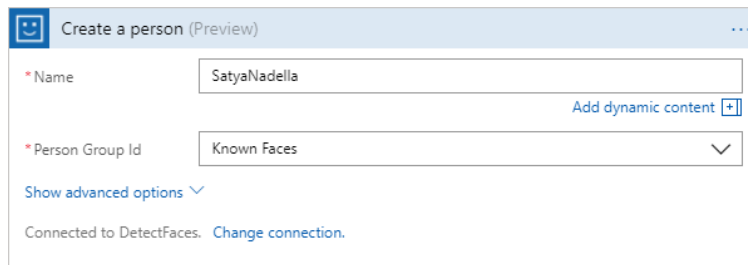


Now that the person group has been created, we will add a new person. This is needed to be able to work with the person group.

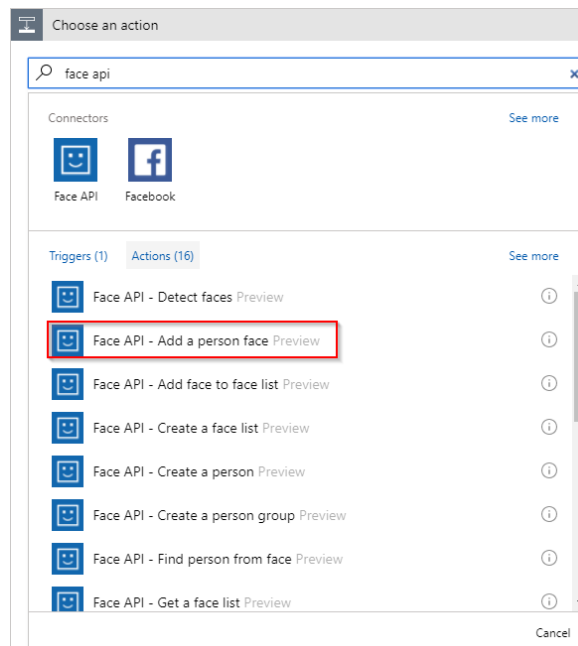
GIBC GLOBAL INTEGRATION BOOTCAMP



As we don't have a person yet, we will add a random person, in this case Satya Nadella.



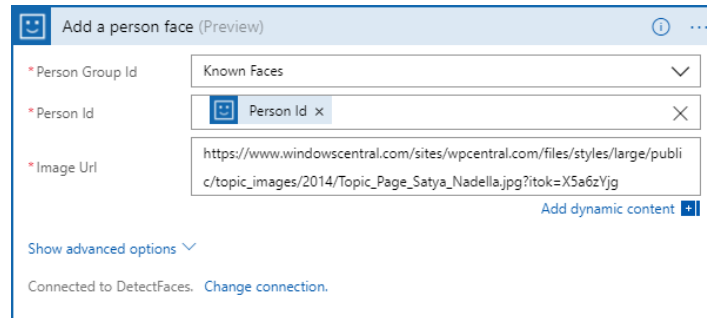
And finally we add a face to our new person.



GBC GLOBAL INTEGRATION BOOTCAMP

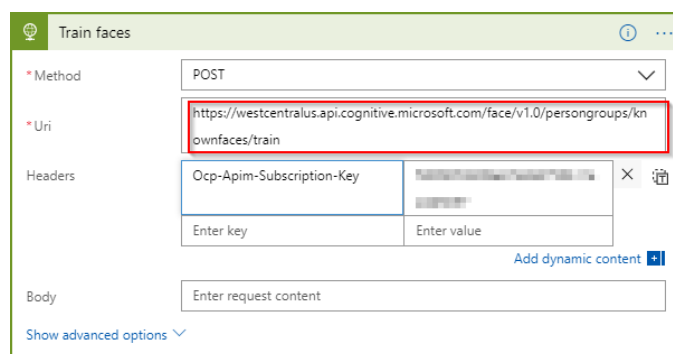
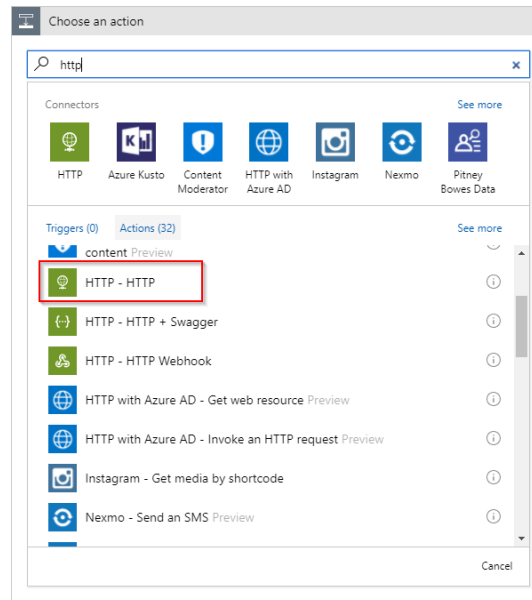
Set the image url to the following:

https://www.windowscentral.com/sites/wpcentral.com/files/styles/large/public/topic_images/2014/Topic_Page_Satya_Nadella.jpg?itok=X5a6zYjg



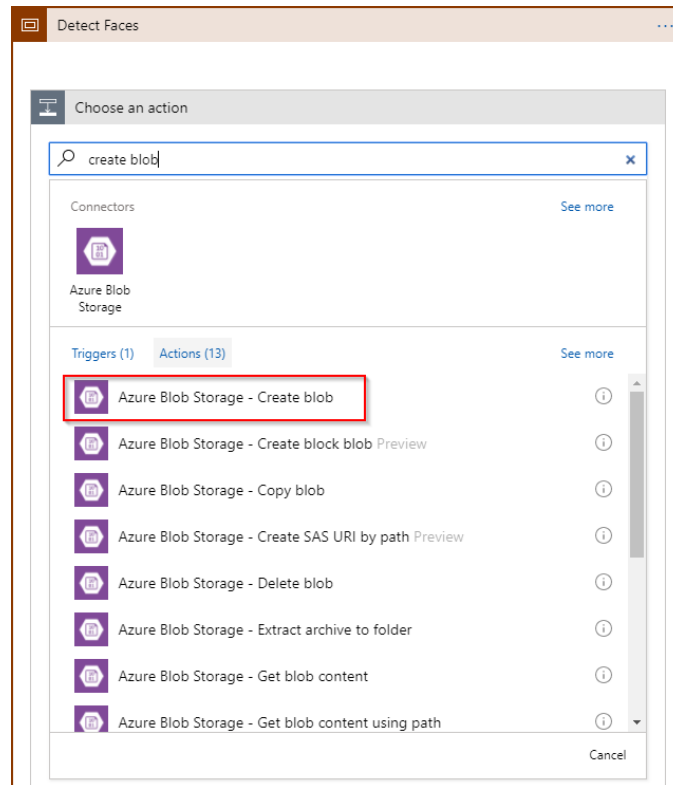
Now that we have created our person group and added a person with a face to it, we need to train the Face API. This makes sure that this person can be recognized in the future. As there is no out of the box action for this in Logic Apps yet, we will call the endpoint

(<https://westcentralus.api.cognitive.microsoft.com/face/v1.0/persongroups/knownfaces/train>) directly from a HTTP action. Make sure to use the endpoint you got while creating your Face API account. The header Ocp-Apim-Subscription-Key should be set to the API Key for your account.

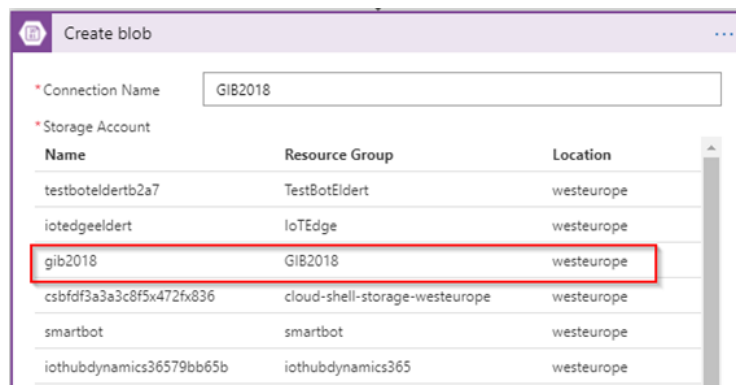


GIBC GLOBAL INTEGRATION BOOTCAMP

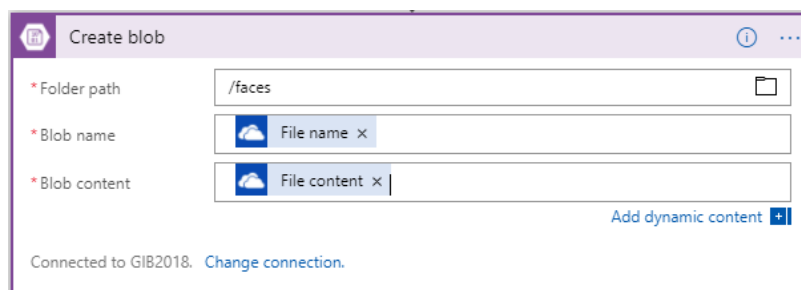
Now add a new scope after the Create person group scope, and name it Check faces. Inside this new scope, create an action to create a blob.



Select the storage account we created earlier on.

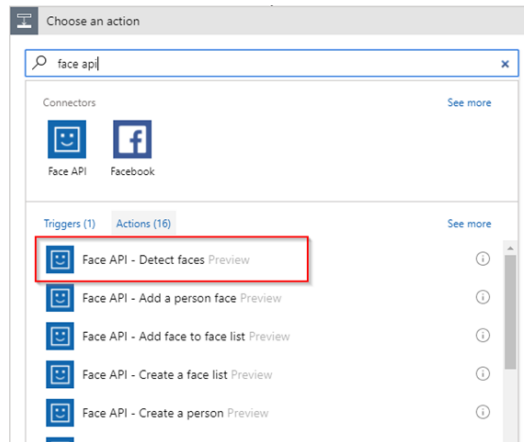


Send the file we received from OneDrive into the storage account.

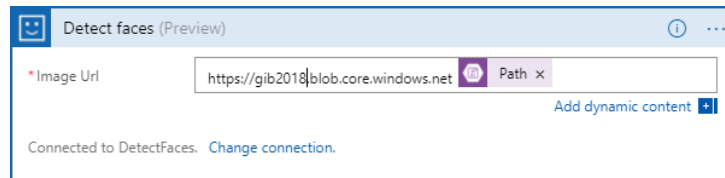


GIBC GLOBAL INTEGRATION BOOTCAMP

We will now use the URL to the blob to detect the face in the Face API.



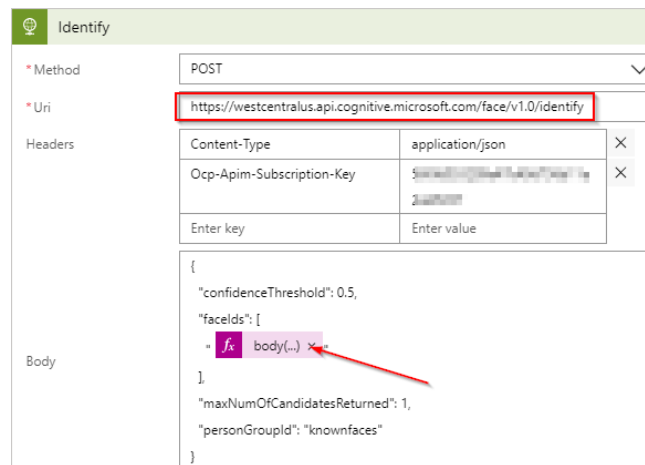
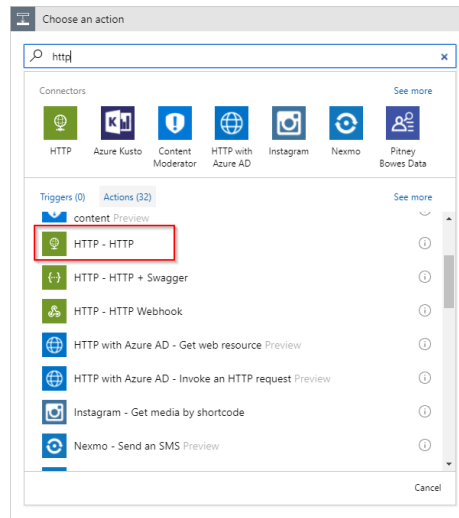
Set the URL to the blob storage (<https://<yourstorageaccountname>.blob.core.windows.net>) as following.



This will give us back a face id, which we can then use to identify the person with this face. Again, we will need to call the URL directly for this using a HTTP action, where the URL in this case is <https://westcentralus.api.cognitive.microsoft.com/face/v1.0/identify>. Use the following JSON as the body. The text in red is an expression which should be set using the expression editor.

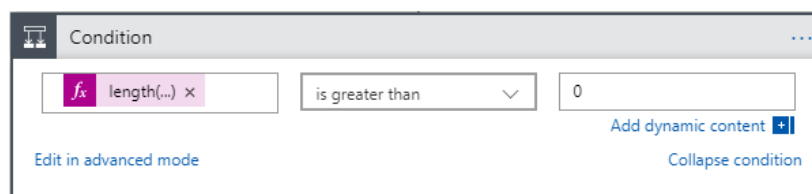
```
{
  "confidenceThreshold": 0.5,
  "faceIds": [
    "body('Detect_faces')[0]['faceId']"
  ],
  "maxNumOfCandidatesReturned": 1,
  "personGroupId": "knownfaces"
}
```

GIBC GLOBAL INTEGRATION BOOTCAMP



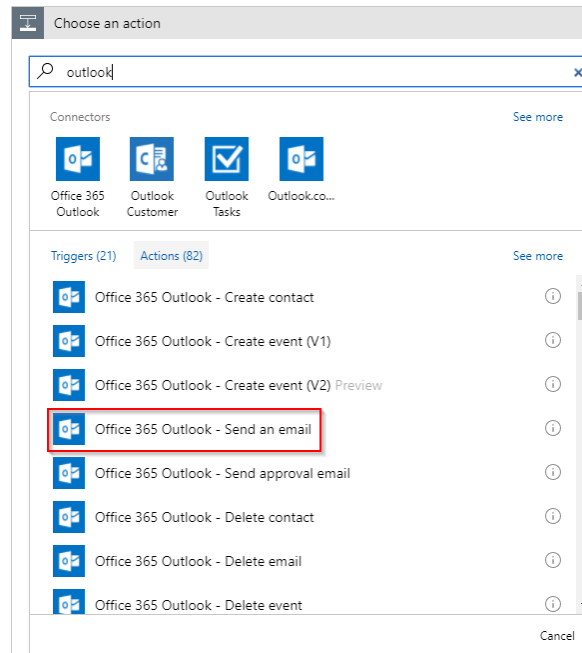
We will now check the response of the identify call by adding a condition to the Logic App. Use the expression editor to set the following expression. The Identify in the expression should be name of the HTTP action used to do the identification.

`length(body('Identify'))[0]?['candidates']`



In the true branch of the condition, which means the person was successfully identified, we will send an email indicating which person was identified.

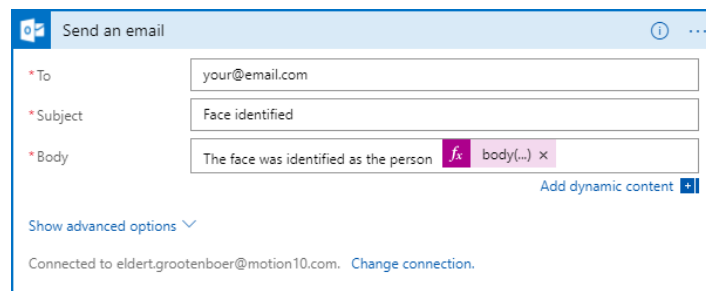
GIBC GLOBAL INTEGRATION BOOTCAMP



Sign in with your Office 365 account (in case you don't have an Office 365 account, use one of the other possibilities to send an email).

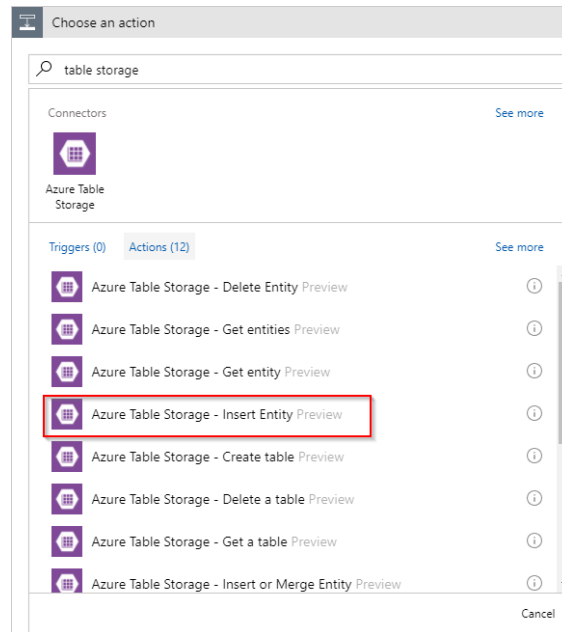
The expression being used here is as following.

`body('Identify')[0]?['candidates'][0]?['personId']`

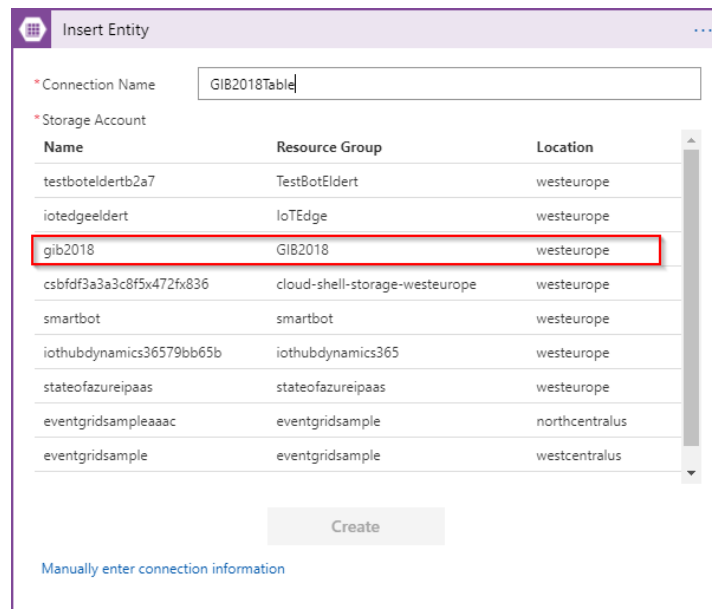


In the false branch, which means the person was not identified against the Face API, we will create the person against the Face API. To do this, we will create an entry in the Table storage we previously created, which can then be used to set the correct person for the uploaded face.

GIBC GLOBAL INTEGRATION BOOTCAMP



Create a new connection against the Table storage we previously created.



Use this JSON to create the entry. The utcNow in red can be set from the expression editor.

```
{
  "PartitionKey": "Faces",
  "RowKey": "utcNow()",
  "ImageURL": "https://gib2018.blob.core.windows.net@{body('Create_blob')}['Path']]",
  "PersonName": "PLEASE SET NAME"
}
```


GIBC GLOBAL INTEGRATION BOOTCAMP

Insert Entity (Preview)

*Table: ReviewFaces

*Entity:

```
{
  "PartitionKey": "Faces",
  "RowKey": "utcNow()",
  "ImageURL": "https://gib2018.blob.core.windows.net/Path",
  "PersonName": "PLEASE SET NAME"
}
```

Add dynamic content

Show advanced options

Connected to GIB2018Table. Change connection.

As we will want to notify the user he has to assign a person to the face, we will send out an email.

Choose an action

Search: outlook

Connectors: Office 365 Outlook, Outlook Customer, Outlook Tasks, Outlook.co...

Triggers (21) | Actions (82)

Office 365 Outlook - Create contact

Office 365 Outlook - Create event (V1)

Office 365 Outlook - Create event (V2) Preview

Office 365 Outlook - Send an email

Office 365 Outlook - Send approval email

Office 365 Outlook - Delete contact

Office 365 Outlook - Delete email

Office 365 Outlook - Delete event

Cancel

Send an email 2

*To: your@email.com

*Subject: Row created in table

*Body: Please assign person name to the row

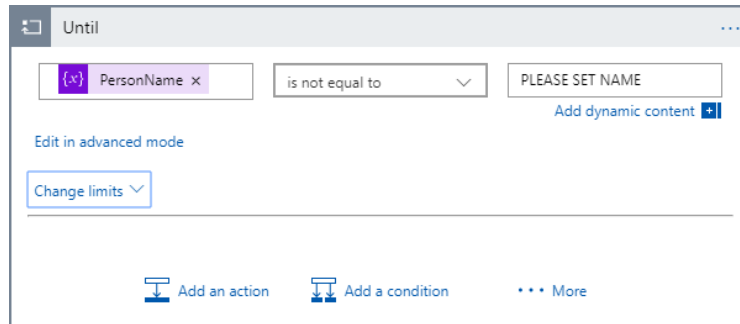
Add dynamic content

Show advanced options

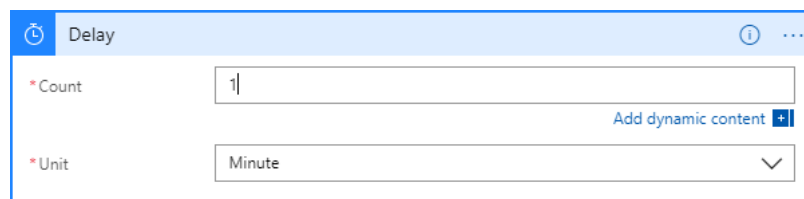
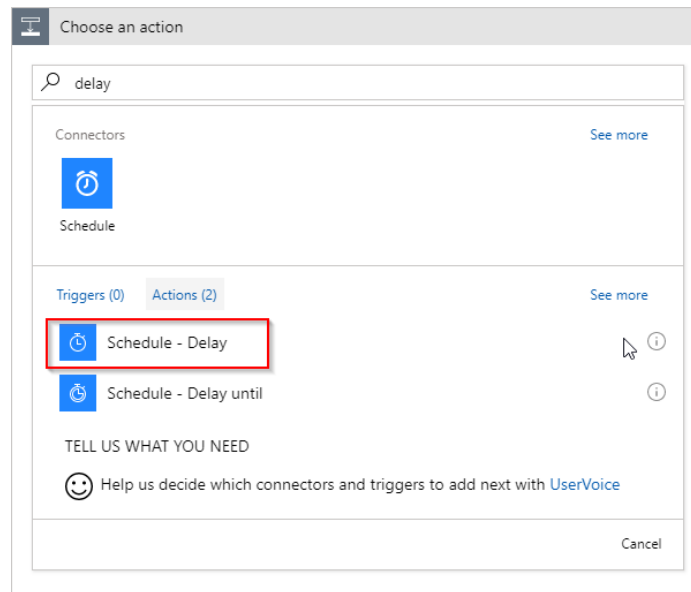
Connected to eldert.grootenboer@motion10.com. Change connection.

Now add a Do until shape to the Logic App, which will be used to check if the name has been set on the row we just created. To do this, we will check the value of the PersonName variable we initially created.

GIBC GLOBAL INTEGRATION BOOTCAMP

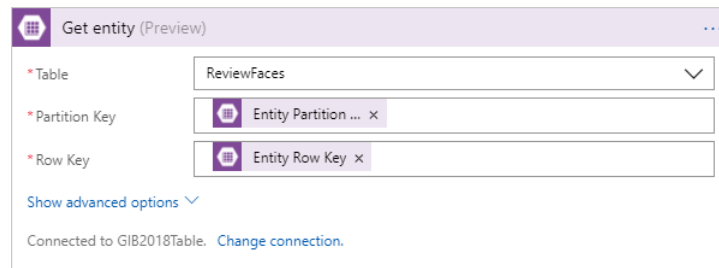
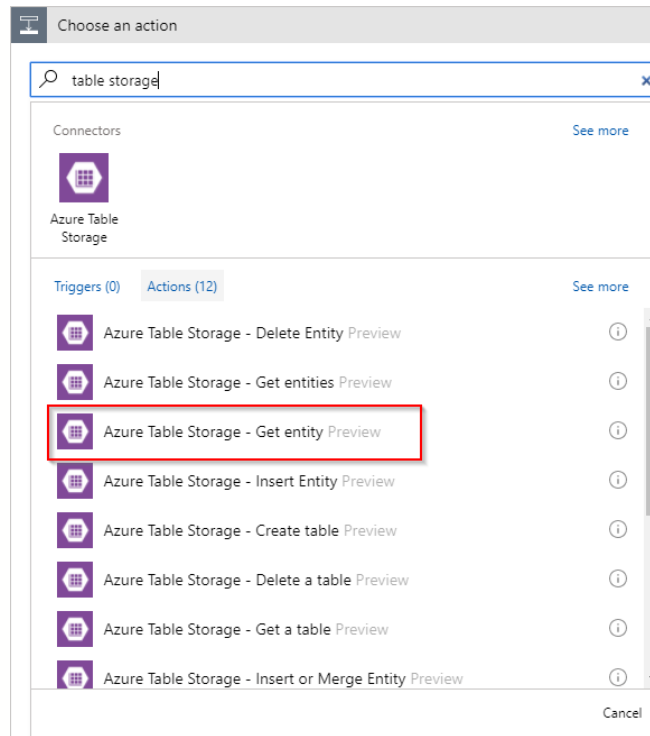


In the Do until shape, add a Delay action, and set it to 1 minute.



Next add a shape to retrieve the entity we just created, to check if it has been updated already.

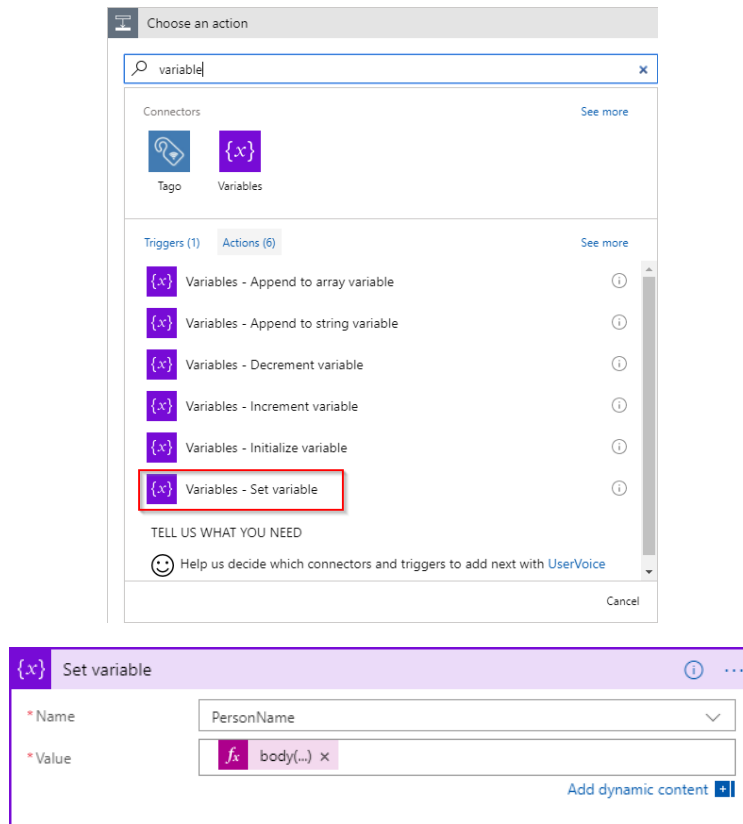
GIBC GLOBAL INTEGRATION BOOTCAMP



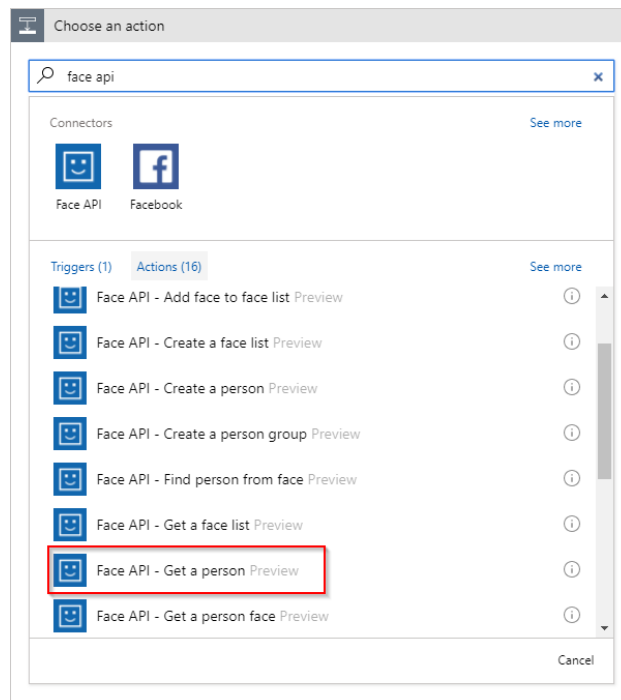
And as a final action inside the Do until shape, update the PersonName variable with the name which is currently set in the table. The expression to be set for this is as following, where the Get_entity name is the name of the previous action.

```
body('Get_entity')['PersonName']
```

GIBC GLOBAL INTEGRATION BOOTCAMP



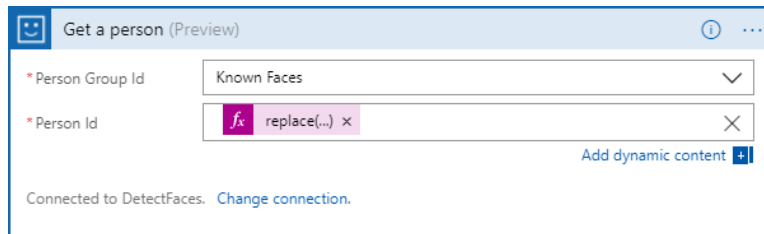
After the Do until shape, add a new action to the person from the Face API corresponding with the name set on the row in the Table storage.



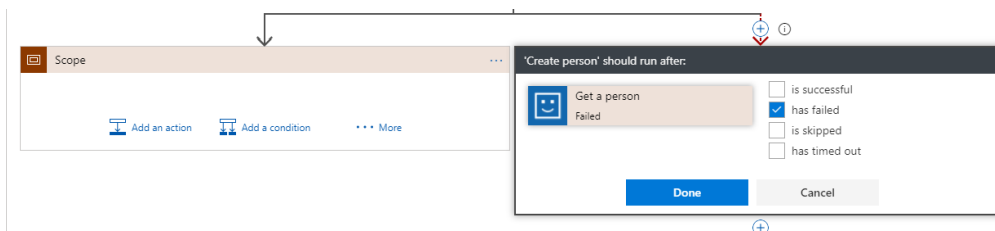
GIBC GLOBAL INTEGRATION BOOTCAMP

The expression that will be used here is the following.

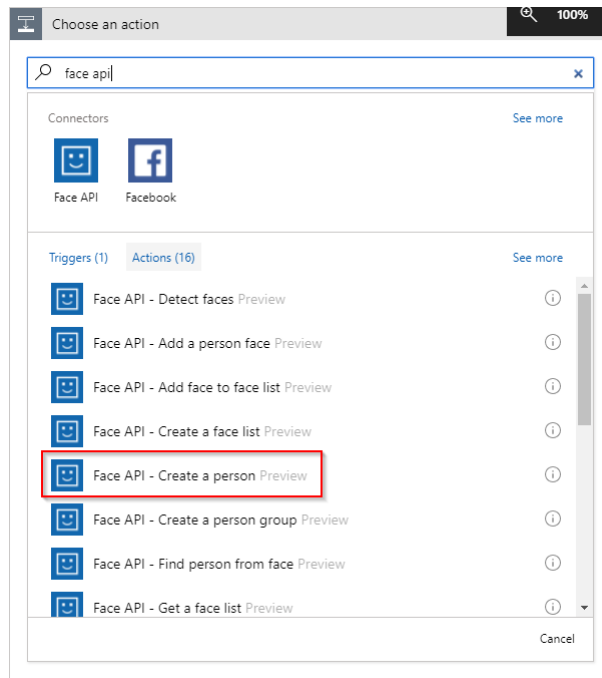
```
replace(variables('PersonName'),' ','')
```



Now add two parallel scopes after the Get person shape, one which will remain empty (in case the person was found), and the second which will create a new person in the Face API (if the person was not found, and thus an error was thrown). The run after for this second scope needs to be set to Has failed, and its name set to Create person.



In the Create person scope we will now create a new person in the Face API.



The expression to be used here is the following.

```
replace(variables('PersonName'),' ','')
```

GIBC GLOBAL INTEGRATION BOOTCAMP

Create a person 2 (Preview)

* Person Group Id: Known Faces

* Name: replace(...)

Add dynamic content

Show advanced options

Connected to DetectFaces. Change connection.

Now after the scopes, add an action which will add the face to the person which was either retrieved or just created.

Choose an action

face api

Connectors: Face API, Facebook

Triggers (1): Face API - Detect faces (Preview)

Actions (16):

- Face API - Add a person face (Preview)
- Face API - Add face to face list (Preview)
- Face API - Create a face list (Preview)
- Face API - Create a person (Preview)
- Face API - Create a person group (Preview)
- Face API - Find person from face (Preview)
- Face API - Get a face list (Preview)

Cancel

Once again, use the following expression.

```
replace(variables('PersonName'), ' ', '')
```

Add a person face 2 (Preview)

* Person Group Id: Known Faces

* Person Id: replace(...)

* Image Url: https://gib2018.blob.core.windows.net Path

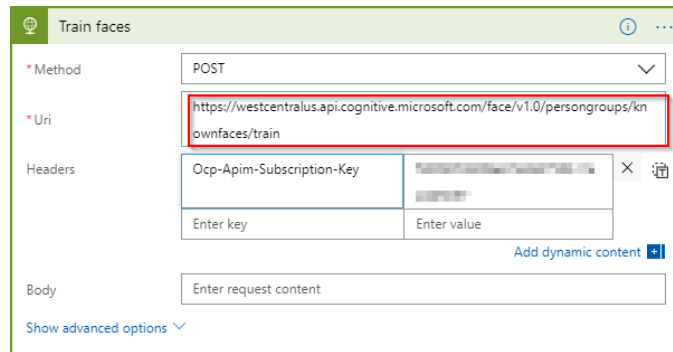
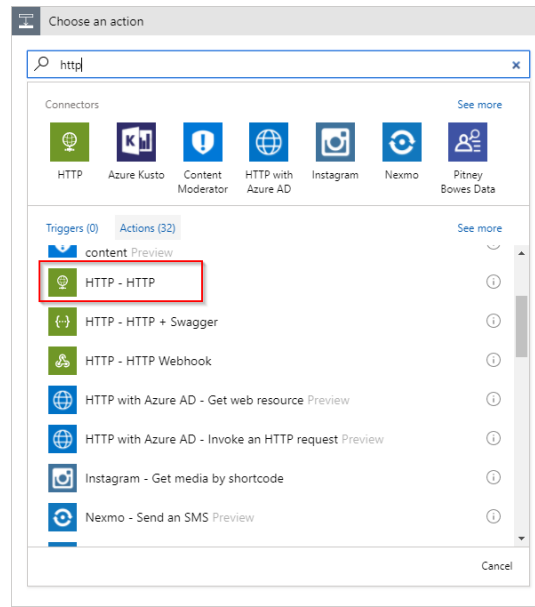
Add dynamic content

Show advanced options

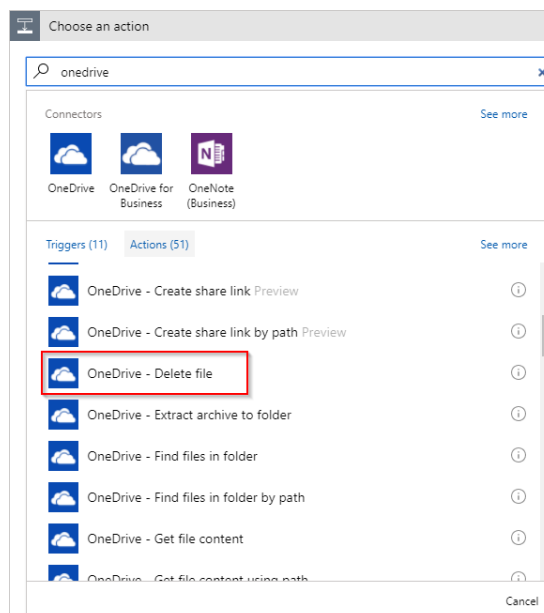
Connected to DetectFaces. Change connection.

And as a final action in this scope, we need to train the Face API with the new face we just added.

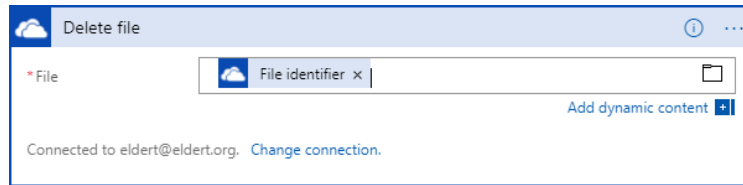
GIBC GLOBAL INTEGRATION BOOTCAMP



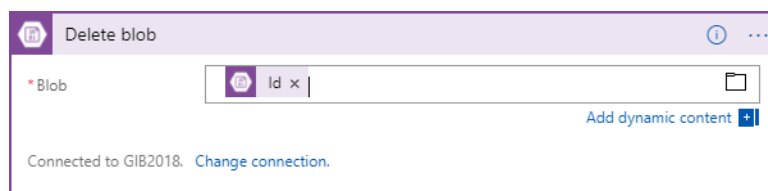
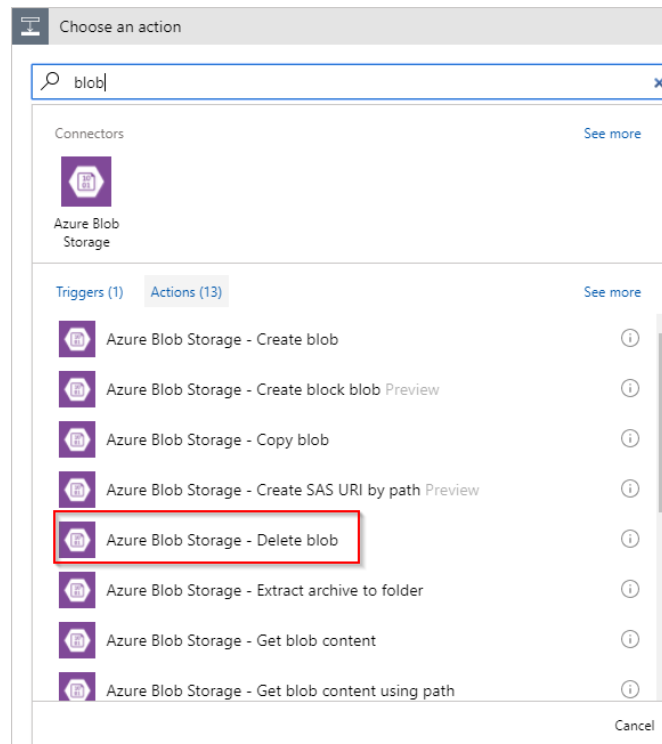
Add a new scope after the Check faces scope, and call this scope Clean up files. Inside this scope, add an action to delete the file we received from OneDrive.



GIBC GLOBAL INTEGRATION BOOTCAMP

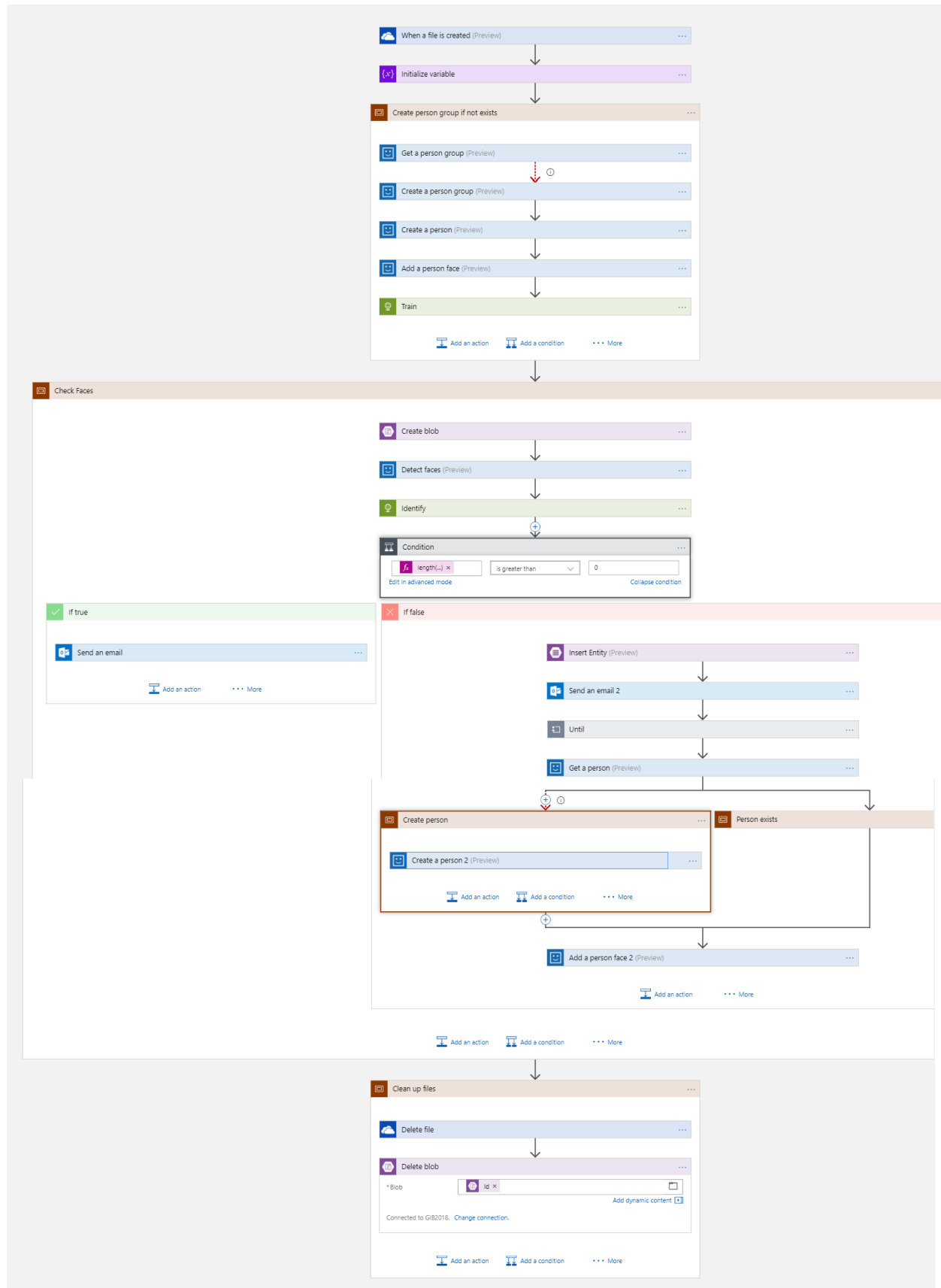


And add an action to delete the file from blob storage as well.



The full solution in the Logic App should now look as following.

GIBC GLOBAL INTEGRATION BOOTCAMP



GIBC GLOBAL INTEGRATION BOOTCAMP

Test the solution

To test the solution we just built, upload a file to the correct folder in your OneDrive. Whenever you upload an image with the face of a new person, a row will be created in the Table storage. To update this row with a person's name, use Azure Storage Explorer to update the row.

