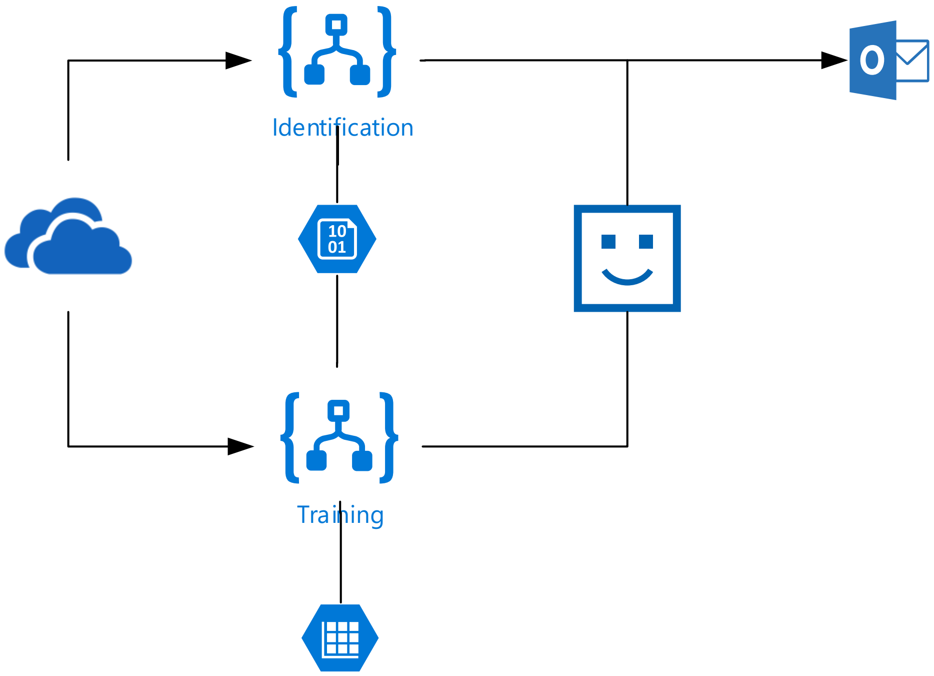
GIBC GLOBAL INTEGRATION BOOTCAMP

LAB: Recognising people with Cognitive Services Face API

Author: Gavin Gregson | Mexia

Objective

In this lab we are going to build a pair of Logic Apps, one to identify people from their photos and one to train the model. Images for both Logic Apps will be picked up from OneDrive, copied to intermediate Blob Storage, and then processed by Cognitive Services Face API.



Prerequisites

- An Azure Subscription
- OneDrive
    - Two folders in OneDrive, an **IdentifyFaces** folder and a **TrainFaces** folder

Recommended

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

Contents

Log in to the Azure Portal

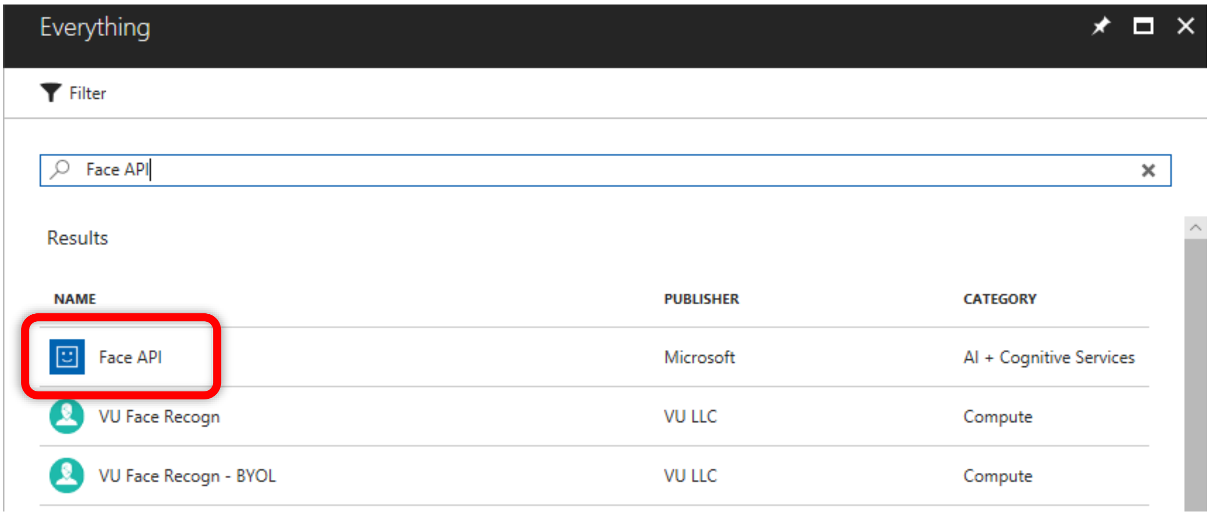
- Log in to https://portal.azure.com

Create a new dashboard

- Create a new dashboard by clicking **+ New dashboard** and name it **GIBC**

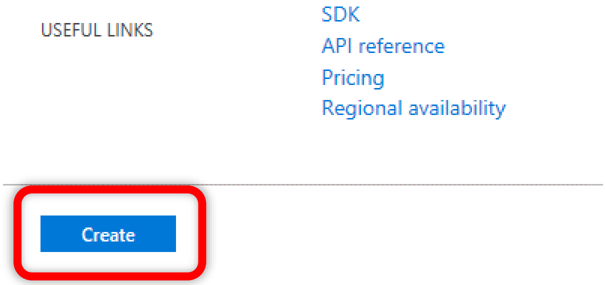
Create a Face API resource and Resource Group

- Click **+ Create a resource**
- Search for **Face API** and select the **Face API** resource from the search results



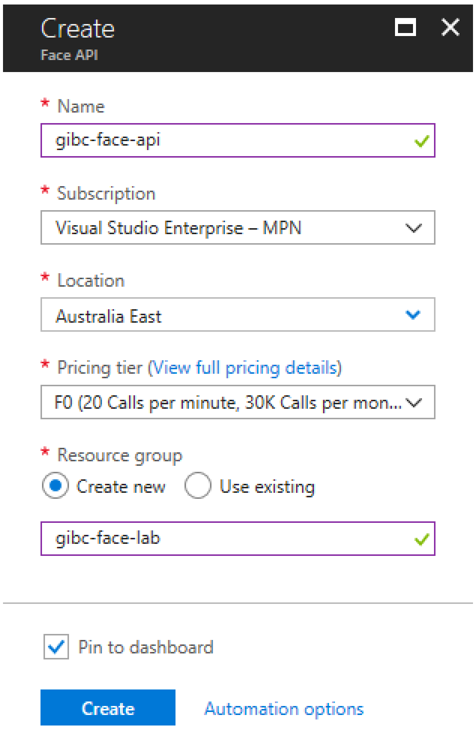
NAME	PUBLISHER	CATEGORY
Face API	Microsoft	AI + Cognitive Services
VU Face Recogn	VU LLC	Compute
VU Face Recogn - BYOL	VU LLC	Compute

- Click the **Create** button at the bottom of the blade that appears



- Name the resource **gibc-face-api,** select a subscription and set the location to **Australia East**
- Set the pricing to the free **F0** tier
    - Note that you can only have 1 F0 tier Face API per subscription
- Under **Resource group** select **Create new** and name the resource group **gibc-face-lab**
- Make sure to tick **Pin to dashboard**

- Your Face API should look like this



Create
Face API

* Name
gibc-face-api ✓

* Subscription
Visual Studio Enterprise – MPN

* Location
Australia East

* Pricing tier (View full pricing details)
F0 (20 Calls per minute, 30K Calls per mon...

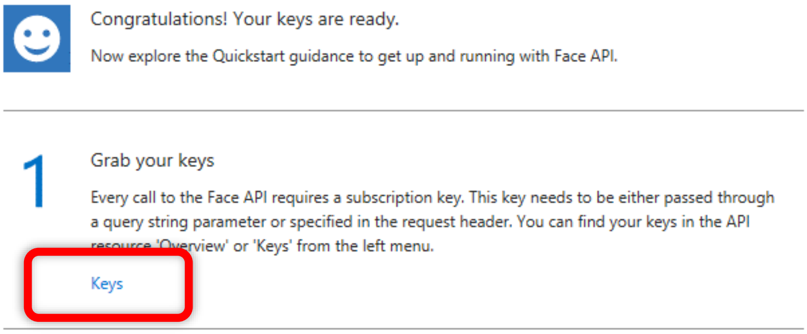
* Resource group
● Create new ○ Use existing
gibc-face-lab ✓

☑ Pin to dashboard

- Click **Create**, deployment should take around 30 seconds
- Click **Go to resource** from the alert that pops up

Get your Face API key

- You'll be taken straight to the **Quick start** page. Click the **Keys** link under **1. Grab your key** and save it somewhere safe for use later



Congratulations! Your keys are ready.

Now explore the Quickstart guidance to get up and running with Face API.

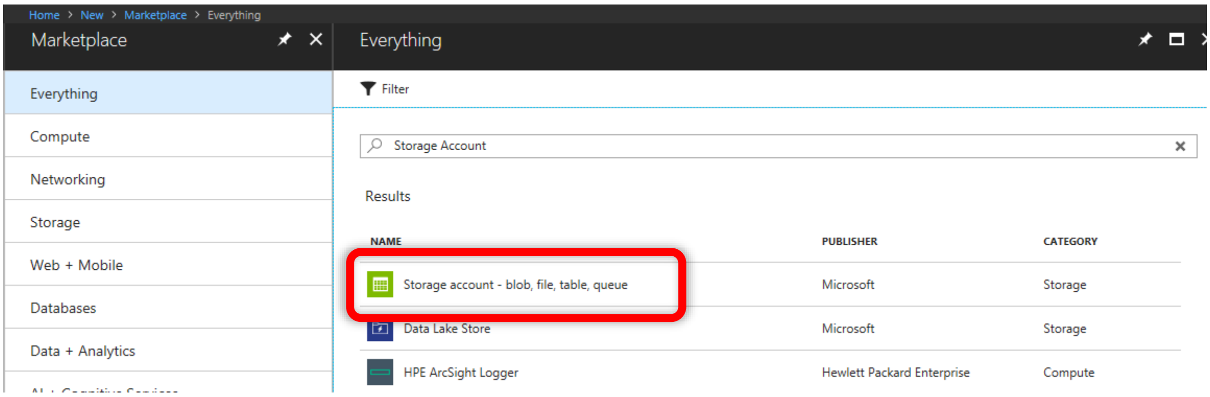
1 Grab your keys

Every call to the Face API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header. You can find your keys in the API resource 'Overview' or 'Keys' from the left menu.

Create Storage Resources

Create a Storage Account

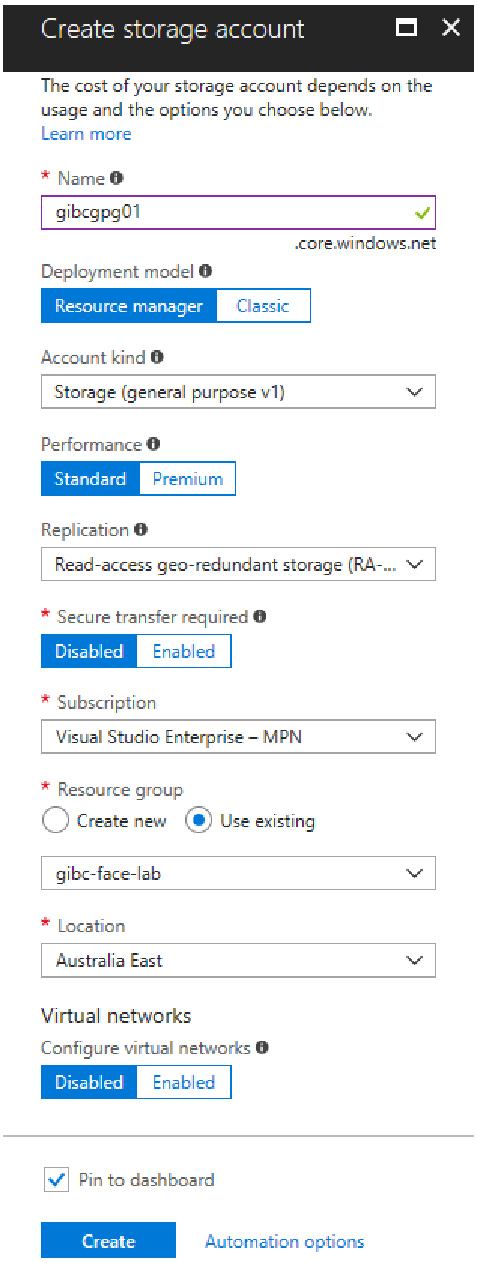
- Click **+ Create a resource** again
- Search for **Storage Account** and select the **Storage account - blob, file, table, queue** resource from the list



NAME	PUBLISHER	CATEGORY
Storage account - blob, file, table, queue	Microsoft	Storage
Data Lake Store	Microsoft	Storage
HPE ArcSight Logger	Hewlett Packard Enterprise	Compute

- Click the **Create** button at the bottom of the blade that appears
- Set a unique name for the storage account (3 to 24 lowercase alphanumeric characters)
- Since this is only for testing you can leave most settings at their defaults:
  - Deployment model: **Resource manager**
  - Account kind: **Storage (general purpose V1)**
  - Performance: **Standard**
  - Replication: **Read-access geo-redundant storage**
  - Secure transfer required: **Disabled**
  - Subscription: ***<your subscription>***
  - Virtual networks: **Disabled**
- For **Resource group** select **Use existing** and select the **gibc-face-lab** resource group that you created when you created the **Face API**
- Make sure to tick **Pin to dashboard**.

- Your Storage Account should look like this



Create storage account

The cost of your storage account depends on the usage and the options you choose below.
Learn more

* Name: gibcgpg01 .core.windows.net

Deployment model: [✓ Resource manager] Classic

Account kind: Storage (general purpose v1)

Performance: [✓ Standard] Premium

Replication: Read-access geo-redundant storage (RA-...

* Secure transfer required: [✓ Disabled] Enabled

* Subscription: Visual Studio Enterprise – MPN

* Resource group: ○ Create new ● Use existing

gibc-face-lab

* Location: Australia East

Virtual networks

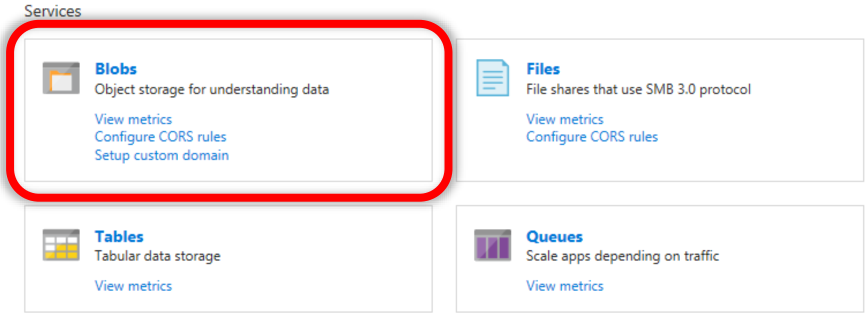
Configure virtual networks: [✓ Disabled] Enabled

☑ Pin to dashboard

- Click **Create**, deployment will take 30 - 60 seconds
- From the alert that pops up click **Go to resource**

Create a Blob Container

- From the Storage Account **Overview** blade click **Blobs**



Services

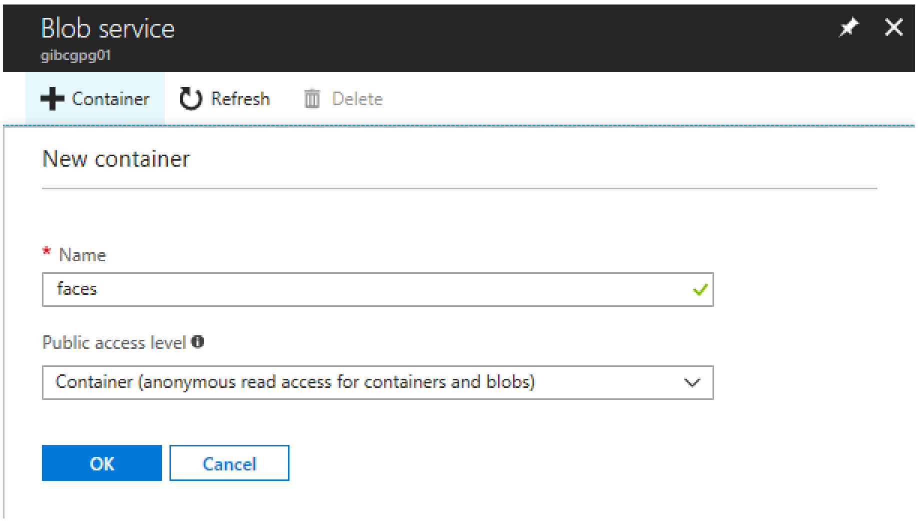
Blobs Object storage for understanding data
View metrics
Configure CORS rules
Setup custom domain

Files File shares that use SMB 3.0 protocol
View metrics
Configure CORS rules

Tables Tabular data storage
View metrics

Queues Scale apps depending on traffic
View metrics

- Copy the **Primary blob service endpoint** (e.g. [https://*your-storage-account-name*.blob.core.windows.net](https://your-storage-account-name.blob.core.windows.net)) somewhere safe for later.
- Click **+ Container**
- Name the new container **faces** and set the **Public access level** to **Container**
    - We're going to be accessing blobs from the container by URL late, and for our lab purposes it's easiest to just make the container public. In a production scenario you would consider using something more secure, such as SAS tokens.
- Your container should look like this



Blob service
gibcgpg01

New container

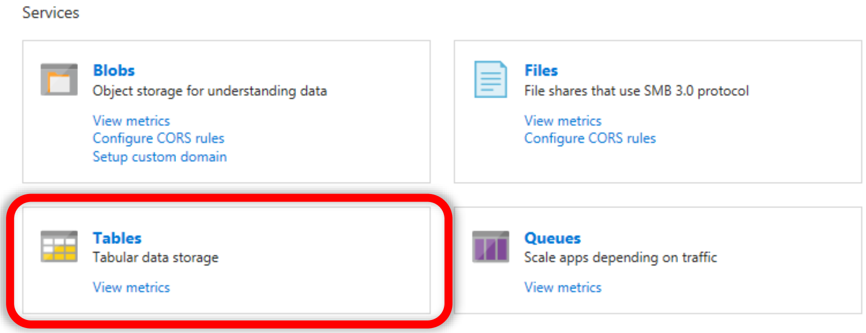
* Name
faces

Public access level
Container (anonymous read access for containers and blobs)

- Click **OK** and close the **Blob service** black using the **X** at the top-right

Create a Table

- You should be back on the Storage Account's **Overview** blade
- Click **Tables**



Services

Blobs
Object storage for understanding data
View metrics
Configure CORS rules
Setup custom domain

Files
File shares that use SMB 3.0 protocol
View metrics
Configure CORS rules

Tables
Tabular data storage
View metrics

Queues
Scale apps depending on traffic
View metrics

- Ignore any prompts about Azure Cosmos DB, we're not going to use that for this lab
- Click **+ Table**
- Set the **Table name** to **PersonIds**

- Your table should look like this

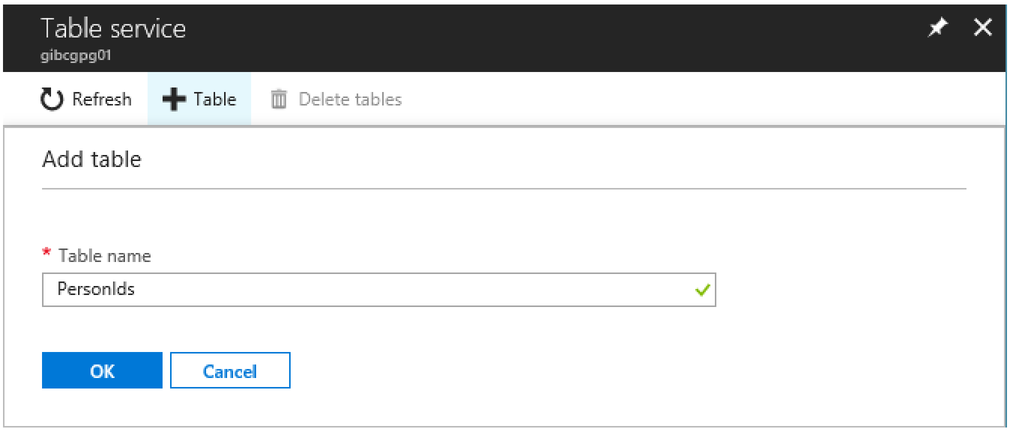


Table service
gibcgpg01

Add table

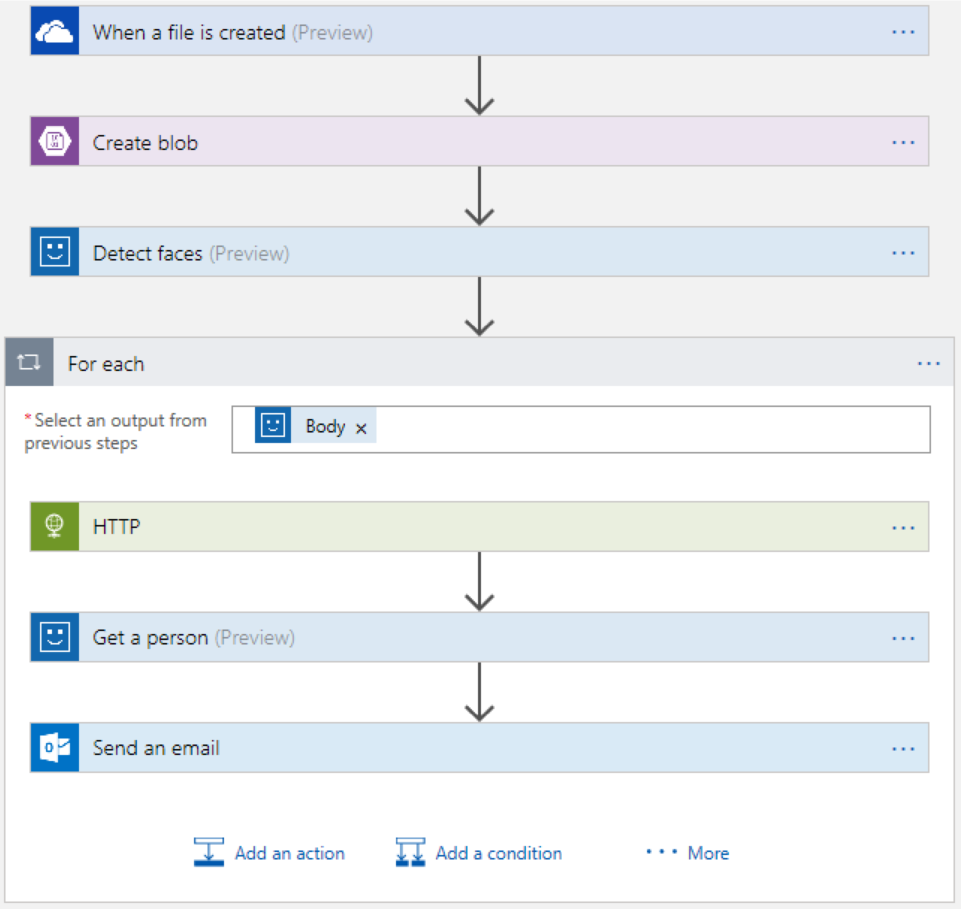
* Table name: PersonIds

- Click **OK**

Identifying a Face with Logic Apps

What We're Going to Build

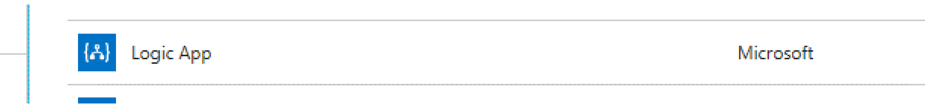
- We're going to build a Logic App which will look like this



Create the Logic App

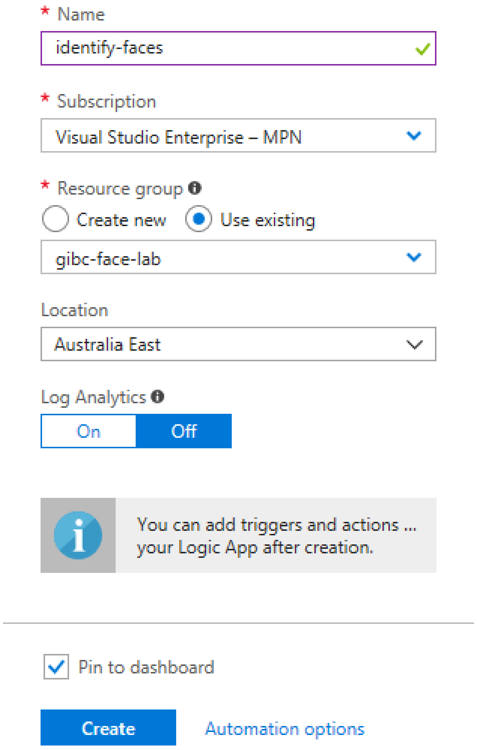
- In the Azure Portal click **+ Create a resource**

- Search for **Logic App** and select the **Logic App** resource from the list



Logic App Microsoft

- Click the **Create** button at the bottom of the blade that appears
- Set the **Name** of the logic app to **identify-faces**
- Set the **Subscription** to your subscription.
- Set **Resource group** to **Use existing** and select the **gibc-face-lab** resource group.
- Set **Location** to **Australia East**.
- Leave **Log Analytics** set to **Off**.
- Make sure you tick **Pin to dashboard**.
- Your Logic App should look like this



* Name
identify-faces ✓

* Subscription
Visual Studio Enterprise – MPN

* Resource group
○ Create new ● Use existing
gibc-face-lab

Location
Australia East

Log Analytics
On [✓ Off]

You can add triggers and actions ... your Logic App after creation.

☑ Pin to dashboard

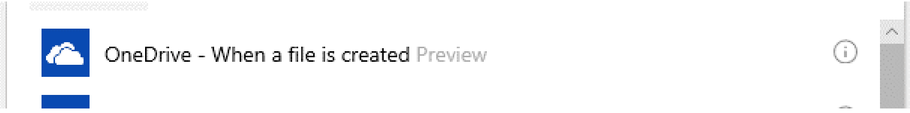
- Click **Create**, deployment should take 30 - 60 seconds
- From the alert that pops up click **Go to resource**

Author the Logic App

- In the **Logic Apps Designer** blade that appears scroll down until you find the **Blank Logic App** template and click it

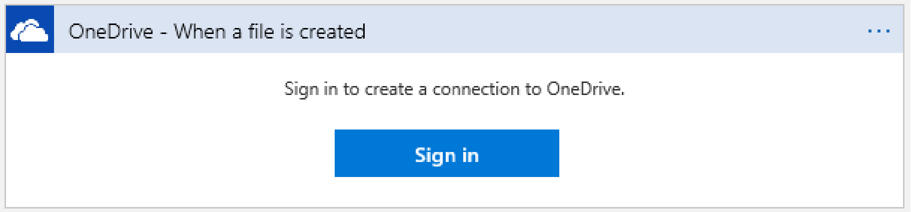
Add the OneDrive Trigger

- The first step is to add a **Trigger**, so in the **Search all connectors and triggers** box search for **OneDrive** and select **OneDrive - When a file is created**



OneDrive - When a file is created Preview

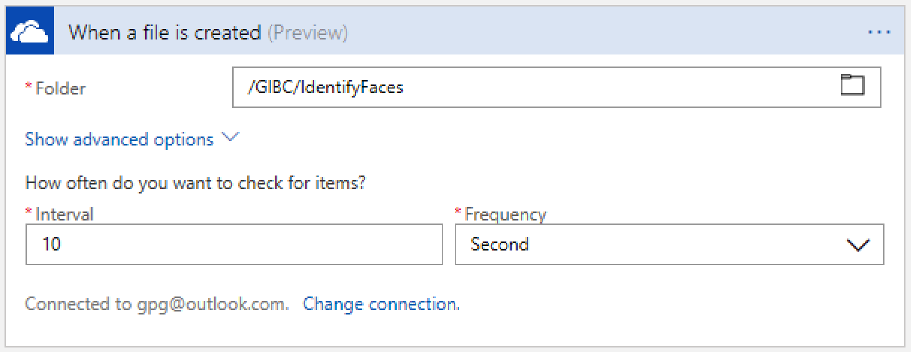
- Since you haven't connected to OneDrive before you'll need to sign in and create a connection, so click the **Sign In** button



OneDrive - When a file is created

Sign in to create a connection to OneDrive.

- Click **Yes** to the permissions request to complete the connection
- In the **Folder** field click the **folder icon** and select the folder in your OneDrive that you'll put faces in for identification (**IdentifyFaces** if you created folders from the recommendations)
- In the action settings, set the **Interval** to **10 Seconds** for this lab so we don't have to wait too long for changes to be detected
- Your action should look like this



When a file is created (Preview)

* Folder: /GIBC/IdentifyFaces

Show advanced options

How often do you want to check for items?

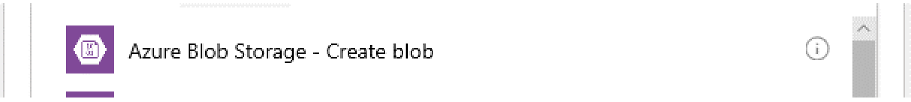
* Interval: 10

* Frequency: Second

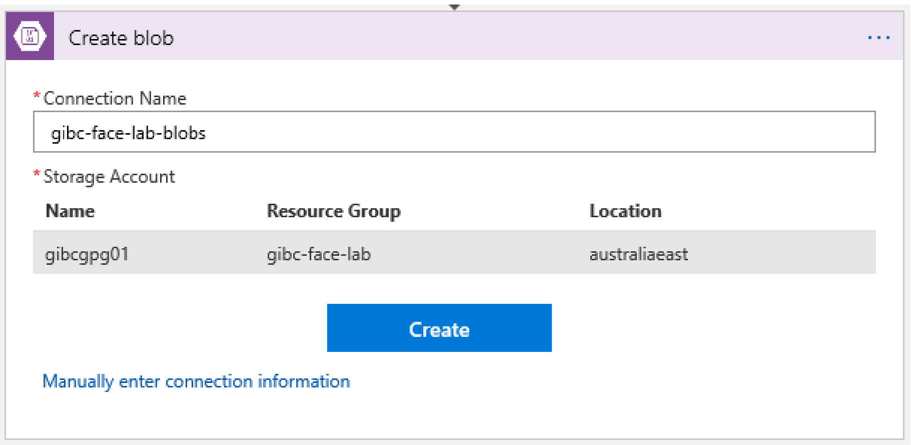
Connected to gpg@outlook.com. Change connection.

Copy the file to Blob storage

- Click **+ New step**, then **Add an action**
- Search for **Blob** and select the **Azure Blob Storage - Create blob** action



- First, we'll have to create a connection to the Storage Account we created earlier
  - Name the connection **gibc-face-lab-blobs**
  - Select the storage account you created earlier
- The connection should look like this



Create blob

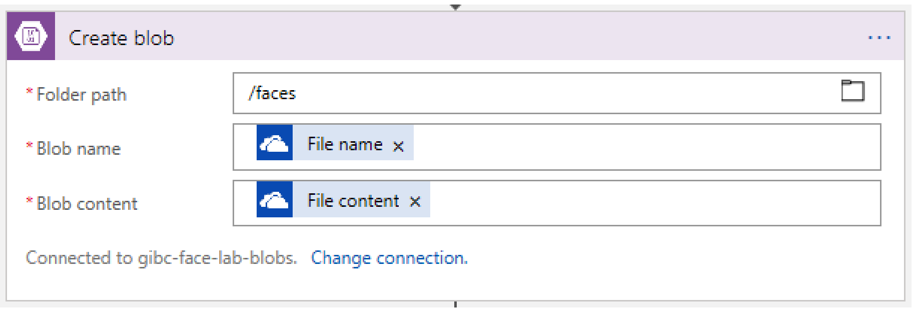
* Connection Name: gibc-face-lab-blobs

* Storage Account

Name	Resource Group	Location
gibcgpg01	gibc-face-lab	australiaeast

- Click **Create**
- In the action settings click the **folder icon** in the **Folder path** field and select the **faces** container you created earlier

- Click in the **Blob name** field and from the popup select **File name**
- Click in the **Blob content** field and from the popup select **File content**
- Your action should look like this



Create blob

* Folder path: /faces

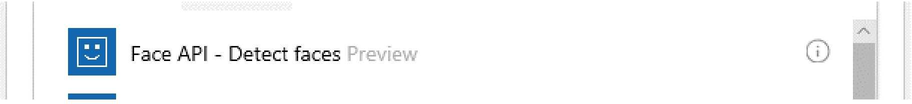
* Blob name: File name ×

* Blob content: File content ×

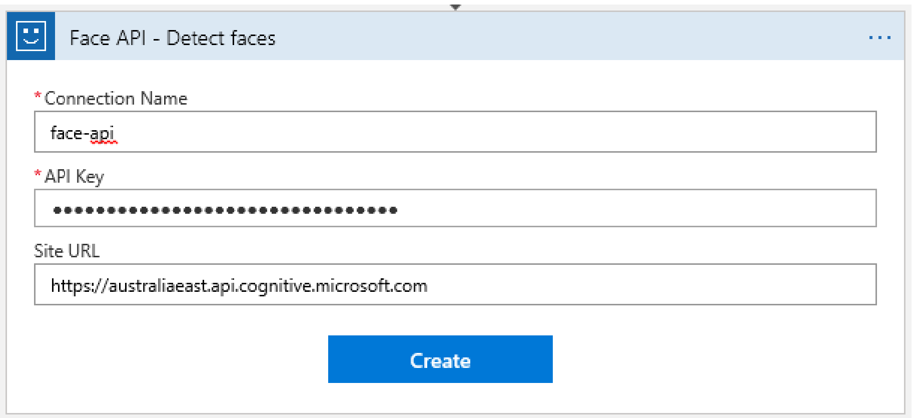
Connected to gibc-face-lab-blobs. Change connection.

Detect any Faces

- Before we can identify a face we first need to detect any faces in the input image
- Click **+ New step**, then **Add an action**
- Search for **Detect Faces** and select the **Face API - Detect faces** action



- We'll need to create a connection to the Face API
- Name the connection **face-api**
- For this GIBC 2018 lab we're initially going to use a pre-trained Face API instance that has already been created, then later you'll create and train your own instance, so for now set the API Key to
  `333fdfc41fcd4e8095b74a644aab72a5`
- Set the **Site URL** to https://australiaeast.api.cognitive.microsoft.com
- Your action should look like this



Face API - Detect faces

* Connection Name: face-api

* API Key: ••••••••••••••••••••••••••••••••

Site URL: https://australiaeast.api.cognitive.microsoft.com

- Click **Create**
- In the action settings click in the **Image Url** field and enter the Blob endpoint you saved earlier without a trailing slash, e.g. https://gibcgpg01.blob.core.windows.net, then with the cursor at the end of the URL click **Path** from the popup

- Your action should look like this

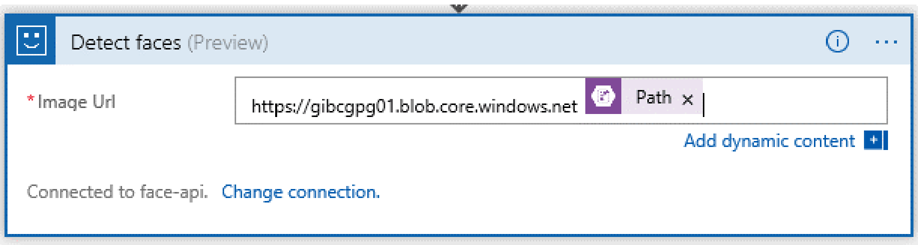

Detect faces (Preview)

*Image Url: https://gibcgpg01.blob.core.windows.net Path ×

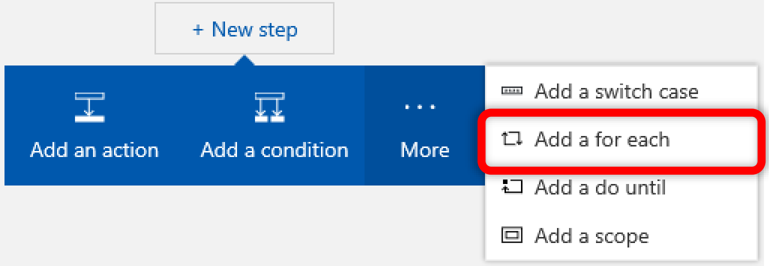
Add dynamic content

Connected to face-api. Change connection.

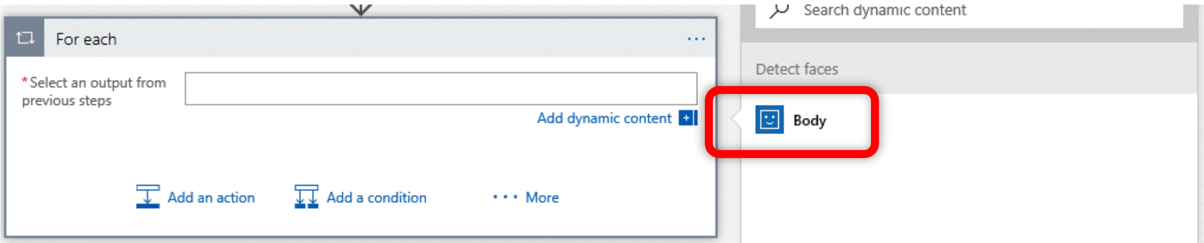
- Note that Detect Faces will return more than one Face ID if the image you provide has multiple faces in it

Identify any Faces

- Now that we have detected a face in the image we need to identify it. Unfortunately, there isn't an action for this yet in Logic Apps, but it's very easy for us to make the REST API call ourselves.
- Because we might have more than one face detected we'll add a **For Each** loop to iterate over each Face ID.
- Click **+ New step,** but this time click **... More** and select **Add a for each**



- Click in the **Select an output from previous steps** field and select **Body** from the popup


For each

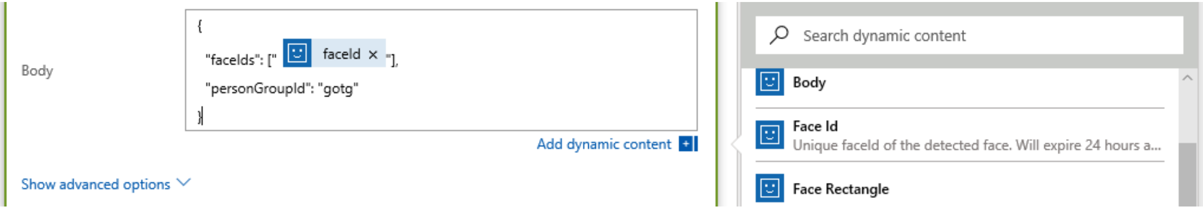
*Select an output from previous steps

- Within the **For each** action click **Add an action**, search for **HTTP**, and select **HTTP – HTTP**

- Set **Method** to **POST**
- Set the **Uri** to https://australiaeast.api.cognitive.microsoft.com/face/v1.0/identify
- Set the first header **key** to **Ocp-Apim-Subscription-Key** and the **value** to **333fdfc41fcd4e8095b74a644aab72a5**
- Set the **Body** to the following JSON content:

```
{
     "faceIds": [""],
     "personGroupId": "gotg"
}
```

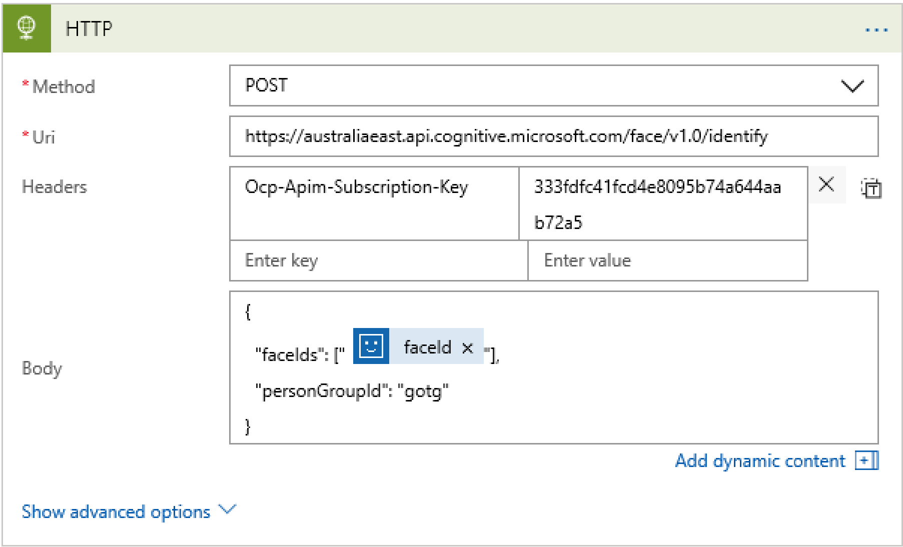
- Put the cursor in between the empty quotes **""** after **"faceIds"** and select **Face Id** from the popup, the **Body** field should look like this



Body

```
{
  "faceIds": [" faceId × "],
  "personGroupId": "gotg"
}
```

- Your action should look like this



HTTP

* Method: POST

* Uri: https://australiaeast.api.cognitive.microsoft.com/face/v1.0/identify

Headers: Ocp-Apim-Subscription-Key — 333fdfc41fcd4e8095b74a644aab72a5

Enter key — Enter value

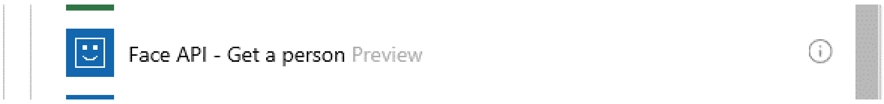
Body

```
{
  "faceIds": [" faceId × "],
  "personGroupId": "gotg"
}
```

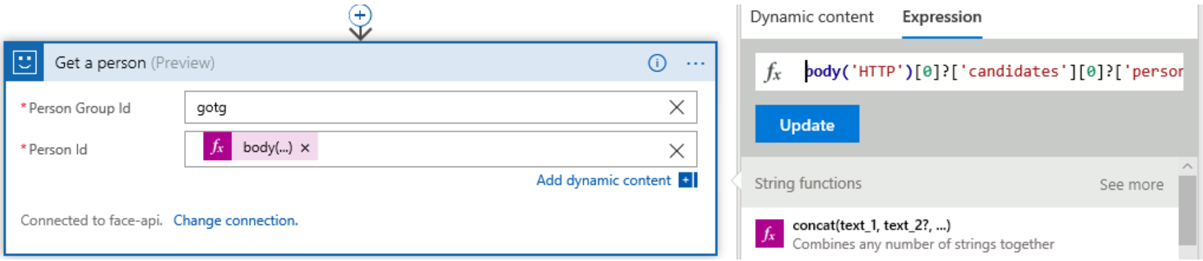
- The response we receive will contain a **personId** for the person that best matches the face provided if the match exceeds a certain threshold (0.5 by default), we will use this to look up the person's details

Get the Identified Person's Name

- In the **For each** loop click **Add an action**, search for **Get Person**, then select the **Face API - Get a person** action



- In the action settings set the **Person Group Id** to **gotg**
- Click in the **Person Id** field, click the **Expression** tab in the popup and set the expression to: **`body('HTTP')[0]['candidates'][0]['personId']`**
- Your action should look like this



Get a person (Preview)

* Person Group Id: gotg

* Person Id: body(...) ×

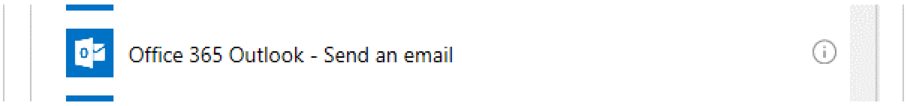
Connected to face-api. Change connection.

```
body('HTTP')[0]?['candidates'][0]?['persor
```

Send an Email with the Persons Name and Picture

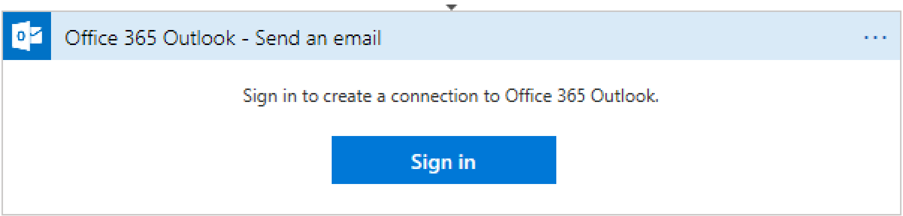
- Finally, we're going to email ourselves when we detect a face successfully. NOTE: This assumes that you have an **Office 365 Outlook** account to use. If you want to use Outlook.com, Gmail, or another service then you can search for the appropriate service and use that connector instead.

- In the **For each** action click **Add an action**, search for **Outlook** and select the **Office 365 Outlook - Send an email** action



Office 365 Outlook - Send an email

- Sign in to your account to create a connection

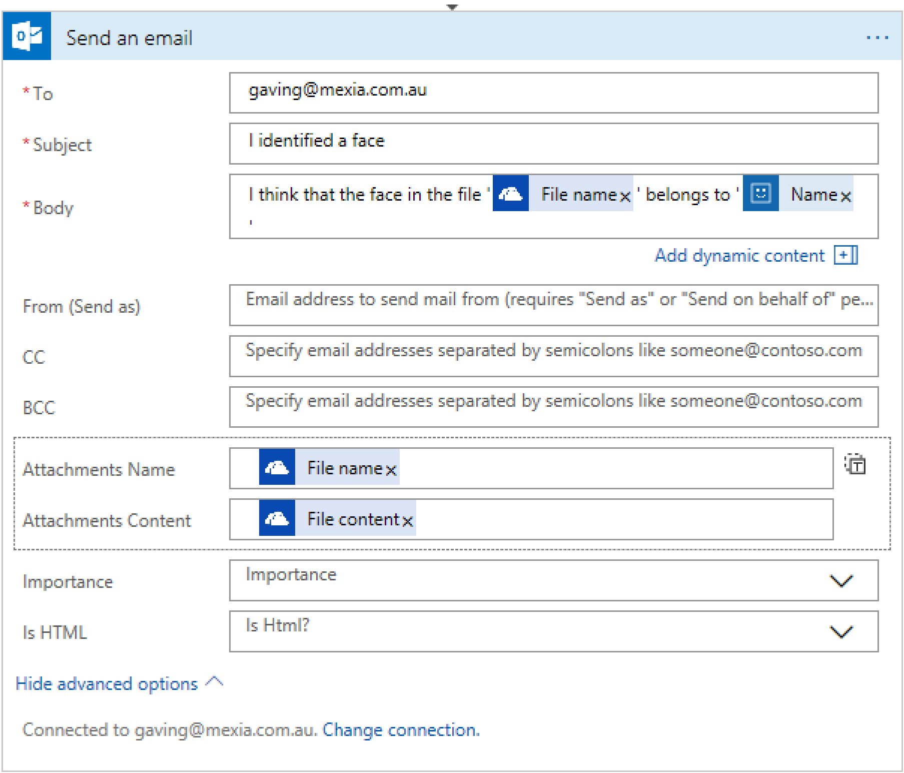


Office 365 Outlook - Send an email

Sign in to create a connection to Office 365 Outlook.

Sign in

- Set **To** to *your email address*
- Set **Subject** to **I identified a face**
- Set **Body** to **I think that the face in the file '' belongs to ''**
- Place the cursor between the empty quotes **''** after the word **file** and select **File name** from the popup under the *When a file is created* category (you'll probably need to scroll to the bottom to find it)
- Place the cursor between the empty quotes **''** after the word **to** and select **Name** from the popup under the *Get a person* category
- Click **Show advanced options**
- Click in the **Attachments Name - 1** field and select **File name** from the popup under the **When a file is created** category.
- Click in the **Attachments Content - 1** field and select **File contents** from the popup, also under the **When a file is created** category.
- Your action should look like this



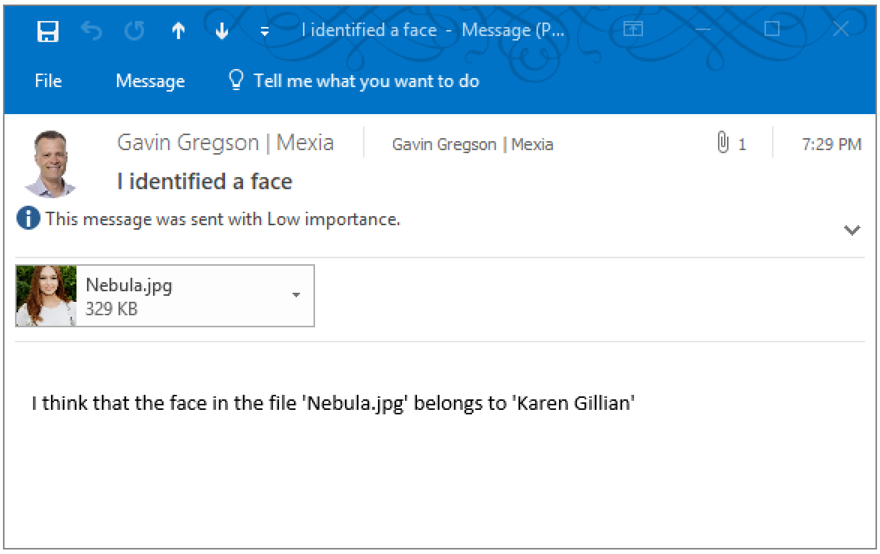
Send an email

* To	gaving@mexia.com.au
* Subject	I identified a face
* Body	I think that the face in the file ' File name × ' belongs to ' Name × '
From (Send as)	Email address to send mail from (requires "Send as" or "Send on behalf of" pe...
CC	Specify email addresses separated by semicolons like someone@contoso.com
BCC	Specify email addresses separated by semicolons like someone@contoso.com
Attachments Name	File name ×
Attachments Content	File content ×
Importance	Importance
Is HTML	Is Html?

Connected to gaving@mexia.com.au. Change connection.

Try it Out

- Drop a photo that the Face API has been trained on into the folder the OneDrive trigger is connected to and if all goes well you should get an email in under 30 seconds with the recognised name and the triggering photo attached, e.g.


I identified a face - Message (P...
File
Message
Tell me what you want to do

Gavin Gregson | Mexia

Gavin Gregson | Mexia

1

7:29 PM

I identified a face

This message was sent with Low importance.

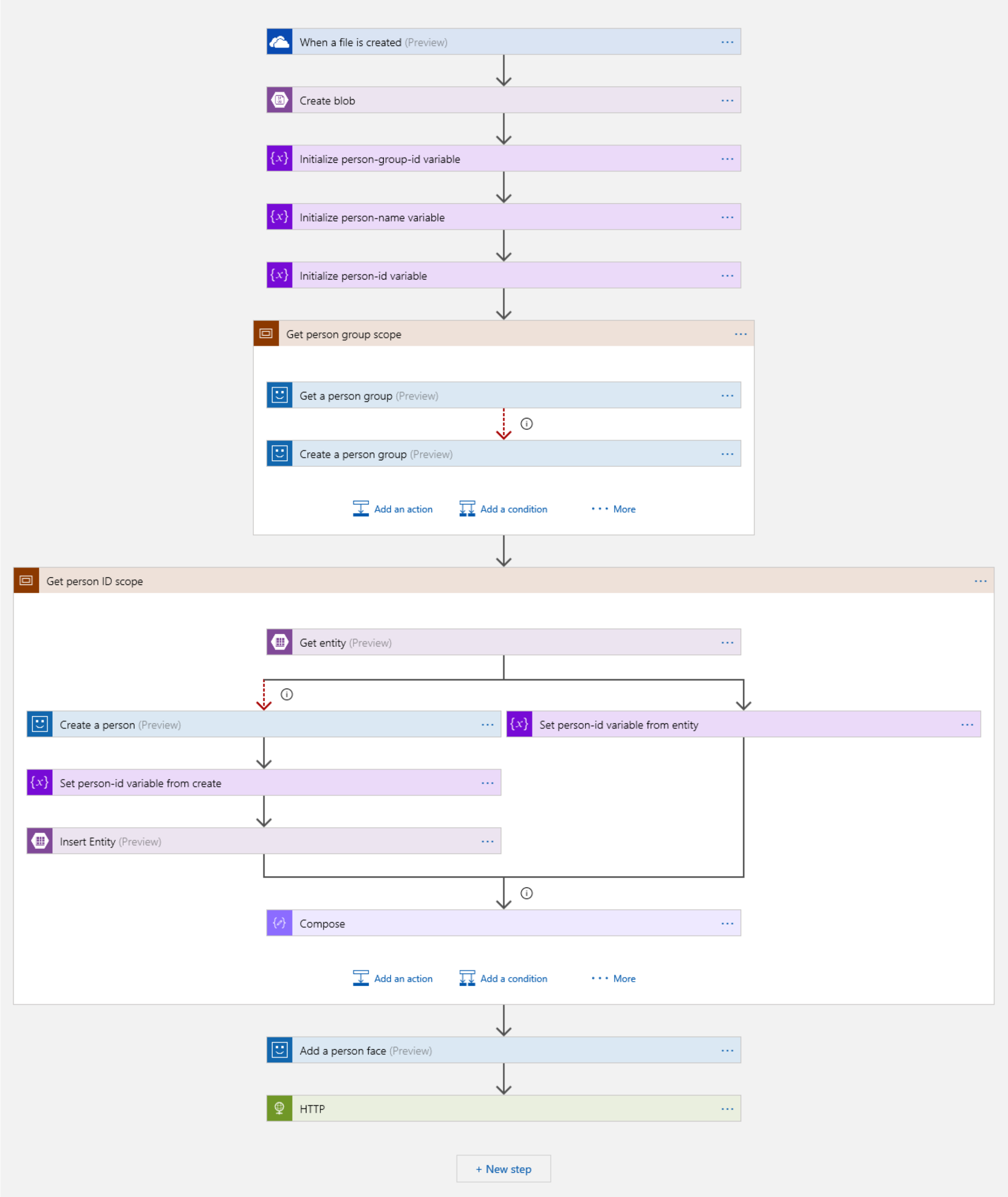
Nebula.jpg
329 KB

I think that the face in the file 'Nebula.jpg' belongs to 'Karen Gillian'

Training the Face API with Logic Apps

What We're Going to Build

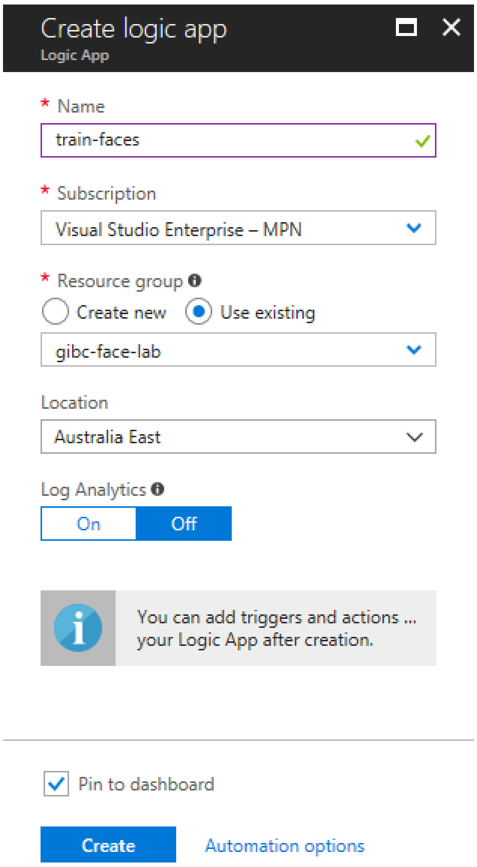
- We're going to build a Logic App which will look like this


When a file is created (Preview)
Create blob
Initialize person-group-id variable
Initialize person-name variable
Initialize person-id variable
Get person group scope
Get a person group (Preview)
Create a person group (Preview)
Add an action
Add a condition
More
Get person ID scope
Get entity (Preview)
Create a person (Preview)
Set person-id variable from entity
Set person-id variable from create
Insert Entity (Preview)
Compose
Add an action
Add a condition
More
Add a person face (Preview)
HTTP
+ New step

Create the Logic App

- In the Azure Portal click **+ Create a resource**
- Search for **Logic App** and select the **Logic App** resource from the list
- Click the **Create** button at the bottom of the blade that appears
- Set the **Name** of the logic app to **train-faces**
- Set the **Subscription** to your subscription.

- Set **Resource group** to **Use existing** and select the **gibc-face-lab** resource group.
- Set **Location** to **Australia East**.
- Leave **Log Analytics** set to **Off**.
- Make sure you tick **Pin to dashboard**.
- Your Logic App should look like this



Create logic app
Logic App

* Name: train-faces

* Subscription: Visual Studio Enterprise – MPN

* Resource group: ○ Create new ● Use existing
gibc-face-lab

Location: Australia East

Log Analytics: On [✓ Off]

You can add triggers and actions ... your Logic App after creation.

☑ Pin to dashboard

- Click **Create**, deployment should take 30 - 60 seconds
- From the alert that pops up click **Go to resource**

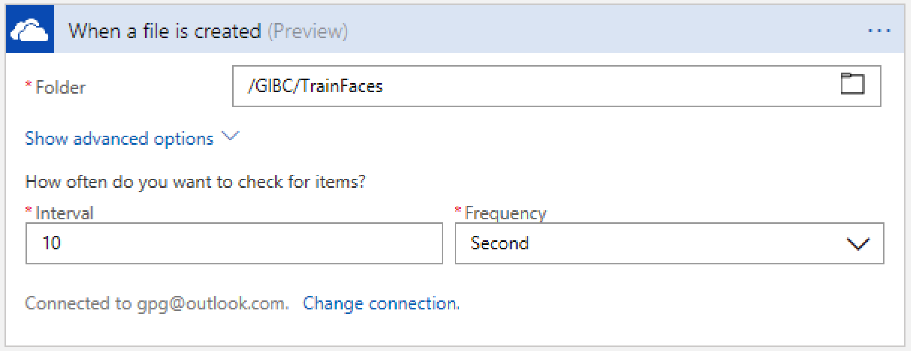
Author the Logic App

- In the **Logic Apps Designer** blade that appears scroll down until you find the **Blank Logic App** template and click it

Add the OneDrive Trigger

- The first step is to add a **Trigger**, so in the **Search all connectors and triggers** box search for **OneDrive** and select **OneDrive - When a file is created**
- You already have a connection to OneDrive so the action should use that automatically, but if you are presented with a Sign In option then just sign in and create a new connection
- In the **Folder** field click the **folder icon** and select the folder in your OneDrive that you'll put faces in for identification (**TrainFaces** if you created folders from the recommendations)
- In the action settings, set the **Interval** to **10 Seconds** for this lab so we don't have to wait too long for changes to be detected

- Your action should look like this


When a file is created (Preview)

*Folder: /GIBC/TrainFaces

Show advanced options

How often do you want to check for items?

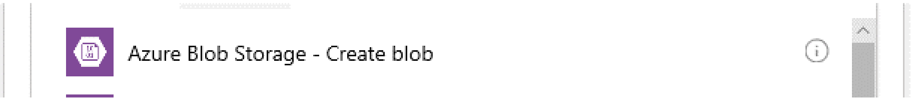
*Interval: 10

*Frequency: Second

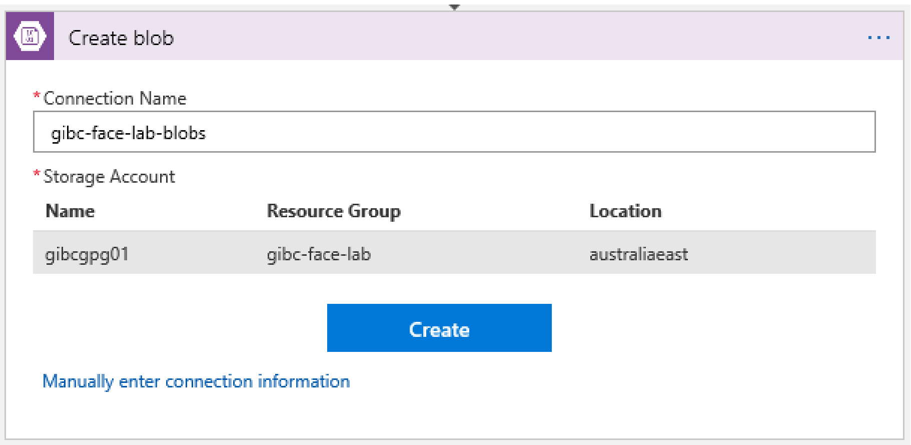
Connected to gpg@outlook.com. Change connection.

Copy the file to Blob storage

- Click **+ New step**, then **Add an action**
- Search for **Blob** and select the **Azure Blob Storage - Create blob** action


Azure Blob Storage - Create blob

- First, we'll have to create a connection to the Storage Account we created earlier
  - Name the connection **gibc-face-lab-blobs**
  - Select the storage account you created earlier
- The connection should look like this

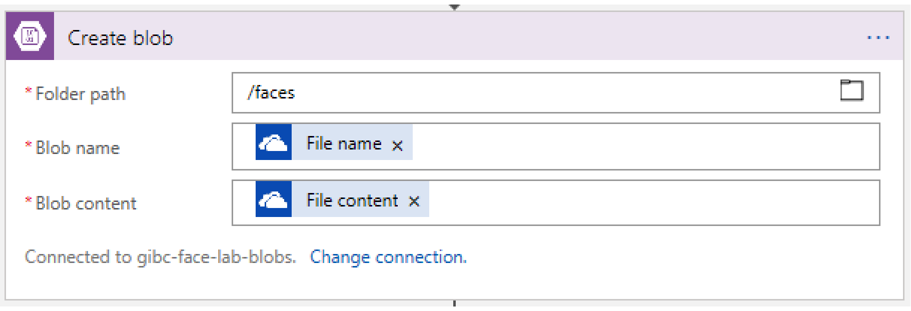

Create blob

*Connection Name: gibc-face-lab-blobs

*Storage Account

Name	Resource Group	Location
gibcgpg01	gibc-face-lab	australiaeast

- Click **Create**
- In the action settings click the **folder icon** in the **Folder path** field and select the **faces** container you created earlier
- Click in the **Blob name** field and from the popup select **File name**
- Click in the **Blob content** field and from the popup select **File content**
- Your action should look like this


Create blob

*Folder path: /faces

*Blob name: File name

*Blob content: File content

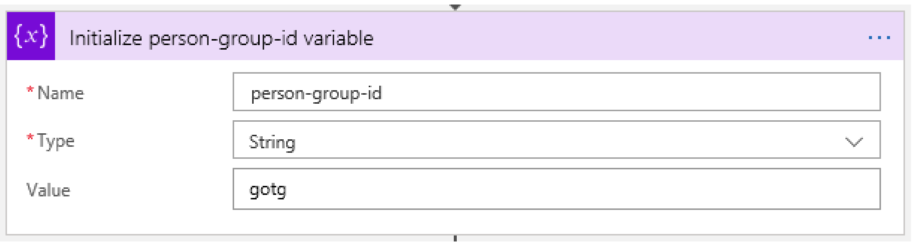
Connected to gibc-face-lab-blobs. Change connection.

Initialise a Variable to store the Person Group ID

- Click **+ New step**, then **Add an action**
- Search for **Variables** and select the **Variables - Initialize variable** action

Variables - Initialize variable

- Set the **Name** to **person-group-id**
- Set the **Type** to **String**
- Set the **Value** to **gotg**
- It's good practice to name actions meaningfully, particularly if there are going to multiple of the same type, so click on the **Ellipsis (...) menu**, select **Rename** and name this Action **Initialize person-group-id variable**
- Your action should look like this



Initialize person-group-id variable

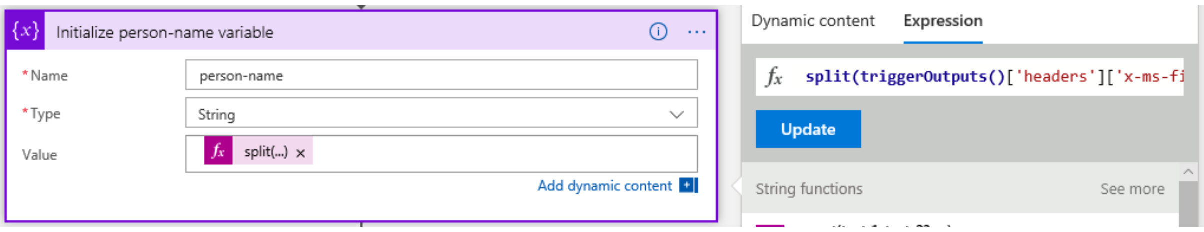
* Name: person-group-id

* Type: String

Value: gotg

Initialise a Variable to store the Person's Name

- As before, Click **+ New step**, then **Add an action**, search for **Variables** and select the **Variables - Initialize variable** action
- Set the **Name** to **person-name**
- Set the **Type** to **String**
- Click in the **Value** field and in the popup click the **Expression** tab
- Set the Expression to
  `split(triggerOutputs()['headers']['x-ms-file-name'], '-')[0]`
- Click on the **Ellipsis (...) menu**, select **Rename** and name this Action **Initialize person-name variable**
- Your action should look like this



Initialize person-name variable

* Name: person-name

* Type: String

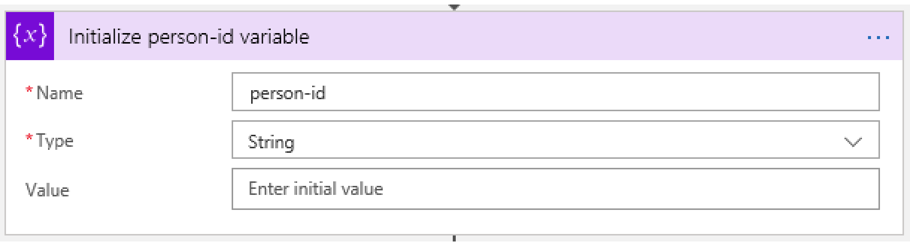
Value: split(...)

split(triggerOutputs()['headers']['x-ms-fi

Initialise a Variable to store the Person's ID

- Once more click **+ New step**, then **Add an action**, search for **Variables** and select the **Variables - Initialize variable** action
- Set the **Name** to **person-id**
- Set the **Type** to **String**
- Leave the **Value** field blank
- Click on the **Ellipsis (...) menu**, select **Rename** and name this Action **Initialize person-id variable**

- Your action should look like this



Initialize person-id variable

* Name: person-id

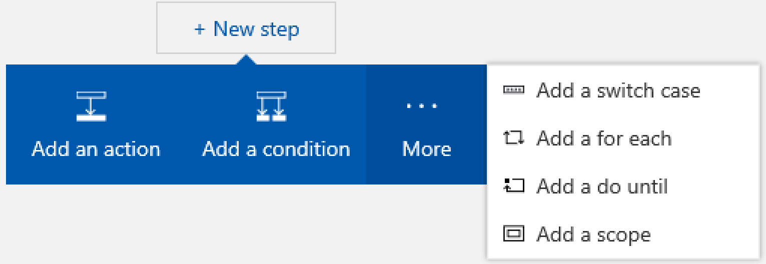
* Type: String

Value: Enter initial value

Get or Create a Person Group

Create a Scope

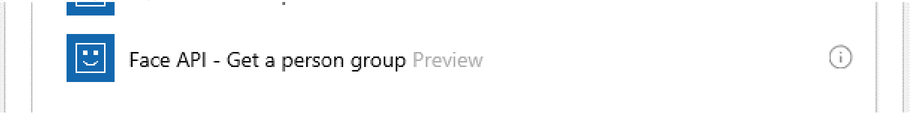
- Add a **scope** by clicking **+ New step**, selecting **... More** and clicking **Add a scope**



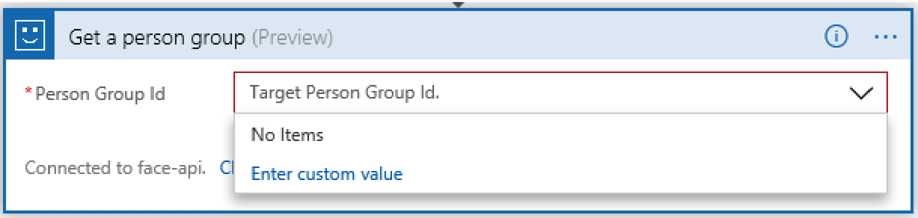
- Click on the **Ellipsis (...) menu**, select **Rename** and name this scope **Get person group scope**
- *Creating a scope just lets us group and organise Actions, and roll them up when we don't need to see them. It has no effect on Variables, in fact you currently cannot initialise a Variable with a scope (but you can set an existing one)*

Try to Get an Existing Person Group

- Within the scope click **Add an action**, search for **person group** and select **Face API - Get a person group**



- Click in the **Person Group Id** field and select **Enter custom value** from the dropdown list



Get a person group (Preview)

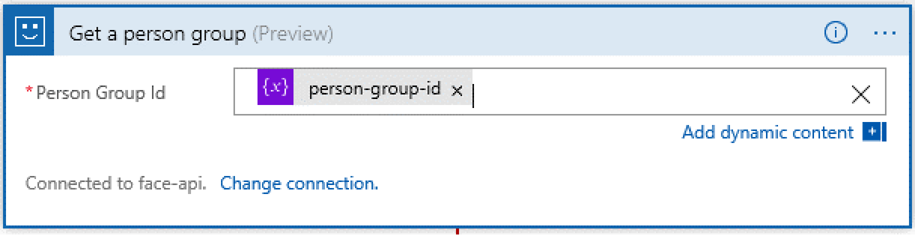
* Person Group Id: Target Person Group Id.

No Items

Enter custom value

Connected to face-api.

- Select the **person-group-id** variable from the popup
- Your action should look like this



Get a person group (Preview)

* Person Group Id: person-group-id

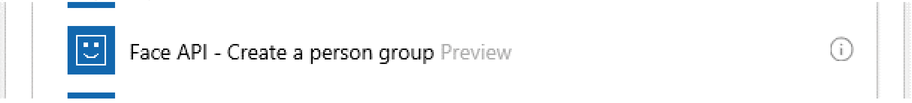
Add dynamic content

Connected to face-api. Change connection.

- The first run through the person group won't exist, so we need to add an action which will run in the event of an error and which will create the missing person group for us.

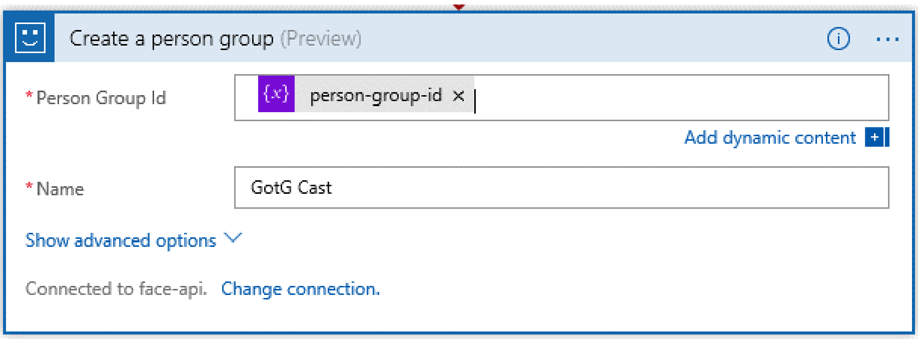
Create a Person Group

- Within the scope click **Add an action**, search for **person group** and select **Face API - Create a person group**



Face API - Create a person group Preview

- Click in the **Person Group Id** field and select the **person-group-id** variable from the popup.
- Set the **Name** field to **GotG Cast**.
- Your action should look like this



Create a person group (Preview)

* Person Group Id: {x} person-group-id ×

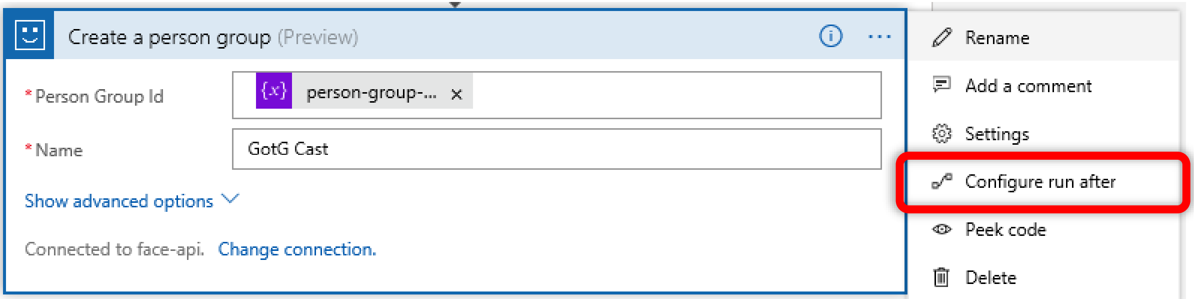
Add dynamic content

* Name: GotG Cast

Show advanced options

Connected to face-api. Change connection.

- You need to configure this Action to run *only* when the previous **Get a person group** Action fails; to do this click the **Ellipsis (...) menu** for the **Create a person group** action and click **Configure run after**



Create a person group (Preview)

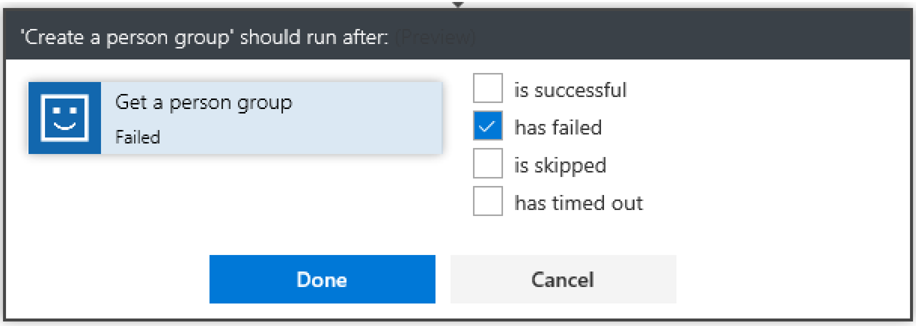
* Person Group Id: {x} person-group-... ×

* Name: GotG Cast

Show advanced options

Connected to face-api. Change connection.

- Untick the **is successful** checkbox and tick the **has failed** checkbox instead, the run after settings should look like this

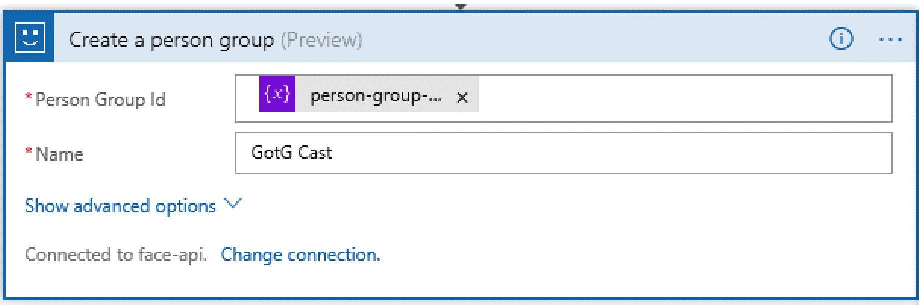


'Create a person group' should run after: (Preview)

Get a person group
Failed

□ is successful
☑ has failed
□ is skipped
□ has timed out

- Click **Done**
- Your action should look like this



Create a person group (Preview)

* Person Group Id: {x} person-group-... ×

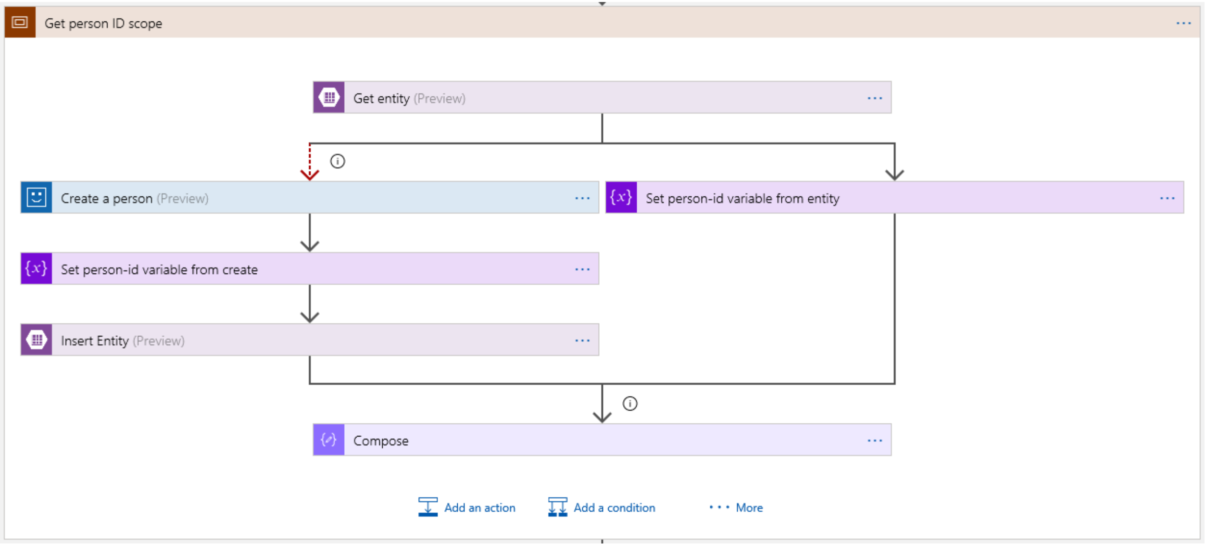
* Name: GotG Cast

Show advanced options

Connected to face-api. Change connection.

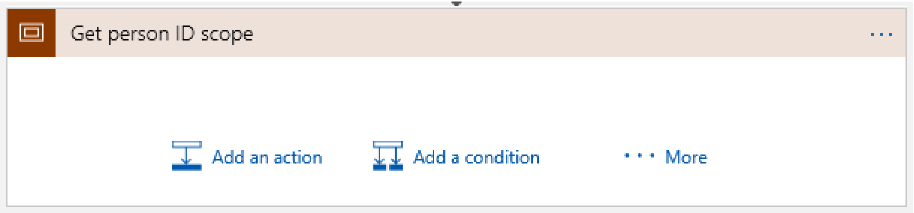
Get or Create a Person ID for a Person

- For simplicity in this lab we're going to encode the name of the person in our filenames by naming convention, e.g. **Amanda Citizen-01.jpg**
- This will be a little more complex, so here is a reference for what we’re going to build in the next few steps



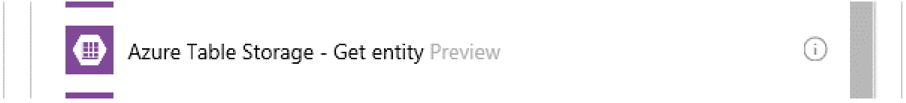
Create a scope

- Add a **scope** by clicking **+ New step**, selecting **... More** and clicking **Add a scope**
- Click on the **Ellipsis (...) menu**, select **Rename** and name this scope **Get person ID scope**
- Your scope should look like this



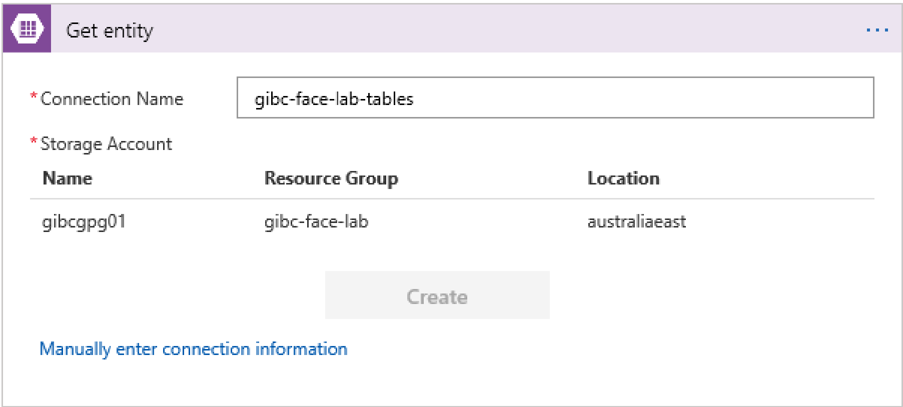
Try to Get an Existing Person ID from Table Storage

- Within the scope click **Add an action**, search for **Table Storage** and select the **Azure Table Storage - Get entity** action



- Since this is the first time you're adding a Table Storage action you'll need to create a connection
- Name the connection **gibc-face-lab-tables** and select the storage account you created for this lab

- Your connection should look like this



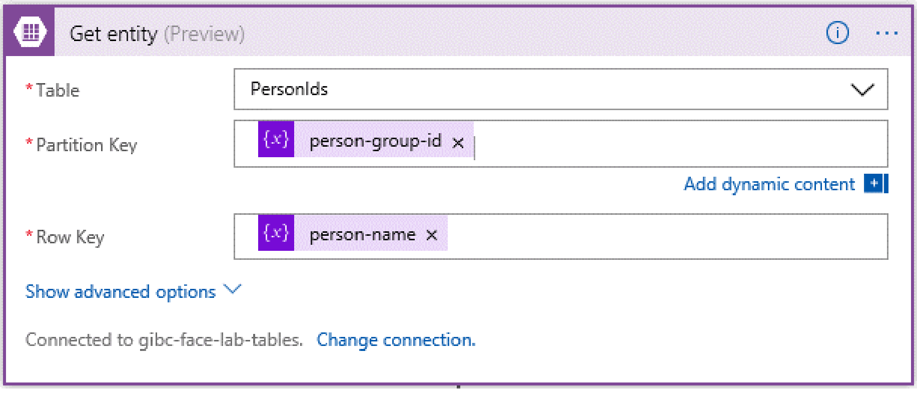
Get entity

* Connection Name: gibc-face-lab-tables

* Storage Account

Name	Resource Group	Location
gibcgpg01	gibc-face-lab	australiaeast

- Click **Create**
- For the **Table** select the **PersonIds** table you created earlier
- Click in the **Partition Key** field and select the **person-group-id** variable from the popup
- Click in the **Row Key** field and select the **person-name** variable from the popup
- Your action should look like this



Get entity (Preview)

* Table: PersonIds

* Partition Key: person-group-id

* Row Key: person-name

Connected to gibc-face-lab-tables. Change connection.

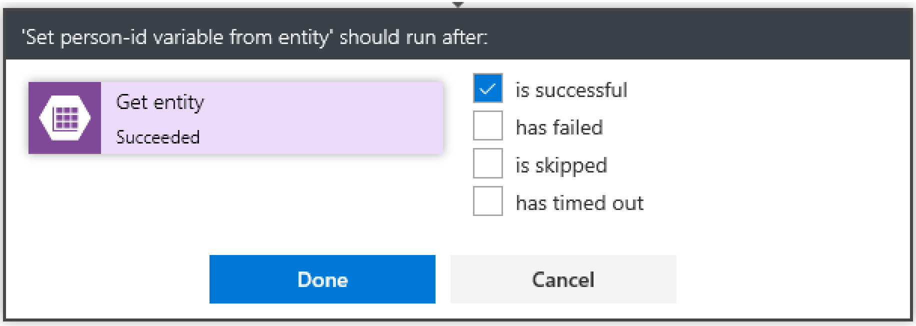
Set the person-id Variable if Successful

- Within the Scope click **Add an action**, search for **Set Variable** and select the **Variables - Set variable** action

Variables - Set variable

- Click in the **Name** field and select the **person-id** variable
- Click in the **Value** field, click the **Expression** tab in the popup and set the expression to **`body('Get_entity')['Id']`**
- Click on the **Ellipsis (...) menu**, select **Rename** and name this Action **Set person-id variable from entity**
- This action should only run if the **Get entity** action succeeded, so click the **Ellipsis (...) menu** again, select **Configure run after**, and ensure **has failed** is unticked (**is successful** should remain ticked)

- The run after settings should look like this

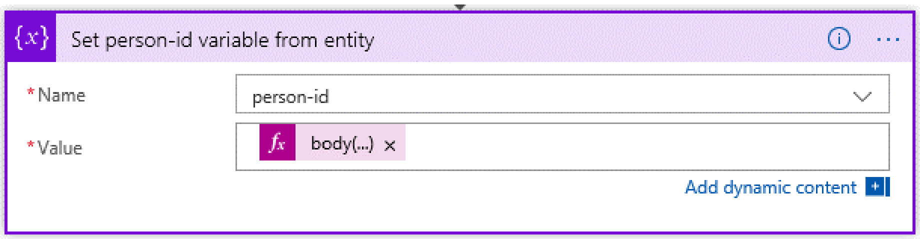


'Set person-id variable from entity' should run after:

Get entity
Succeeded

☑ is successful
☐ has failed
☐ is skipped
☐ has timed out

- Click **Done**
- Your action should look like this



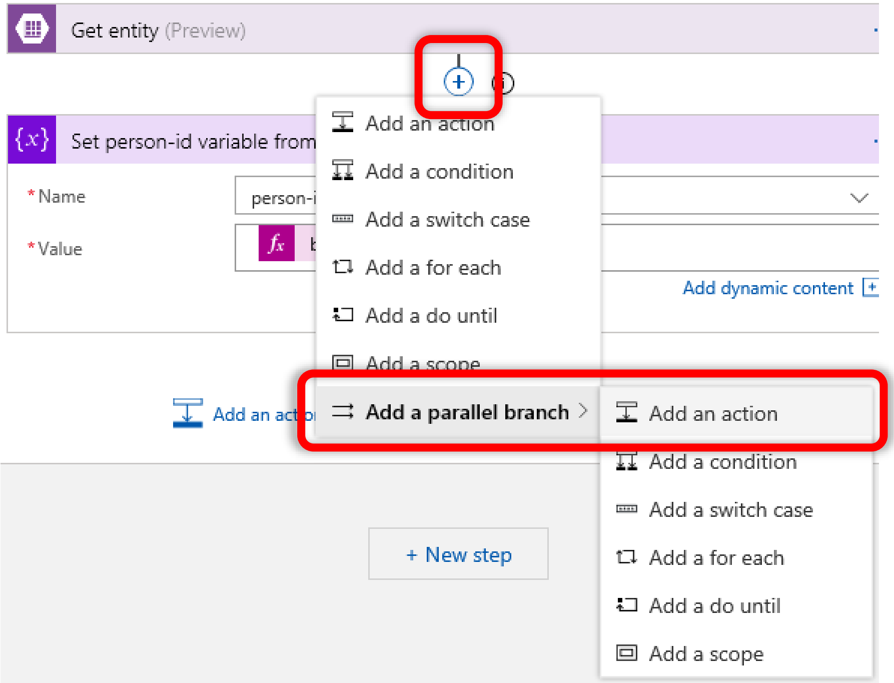
Set person-id variable from entity

*Name: person-id

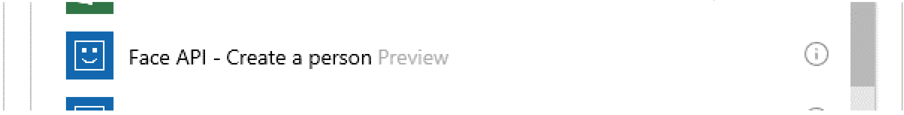
*Value: body(...)

Create a Person ID if None Exists

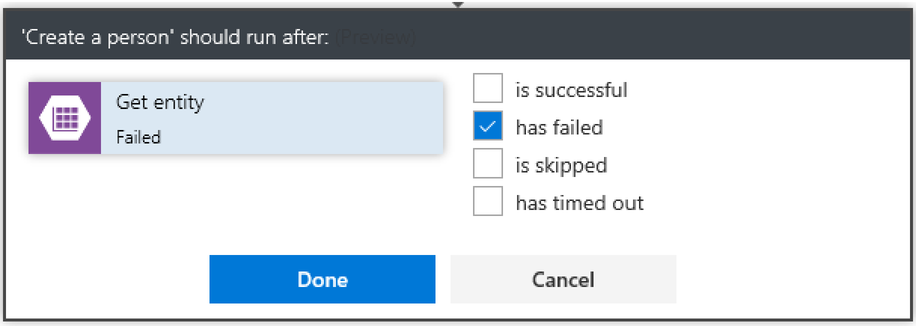
- Now we have a person ID if the person already exists, but we need to handle the case where they don't already exist
- If there's no matching record in Table Storage the **Get entity** action will error, so we'll add a separate branch to execute in that case
- Hover over the arrow between the **Get entity** and click the **+** sign, select **Add a parellel branch**, then **Add an action**



- Search for **Create Person** and select the **Face API - Create a person** action



- Click in the **Person Group Id** field and select **Enter custom value** and select the **person-group-id** variable from the popup.
- Click in the Name field and select the **person-name** variable from the popup
- This action should only run if the **Get entity** action failed, so click the **Ellipsis (...) menu** again, select **Configure run after**, and untick **is successful** and tick **has failed**
- The **Create a person** action will now run only when the **Get entity** Action fails
- The run after settings should look like this

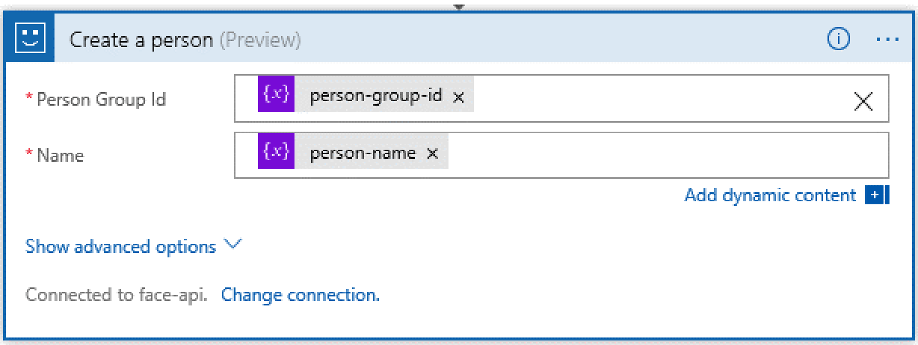


'Create a person' should run after: (Preview)

Get entity
Failed

□ is successful
☑ has failed
□ is skipped
□ has timed out

- Click **Done**
- Your action should look like this



Create a person (Preview)

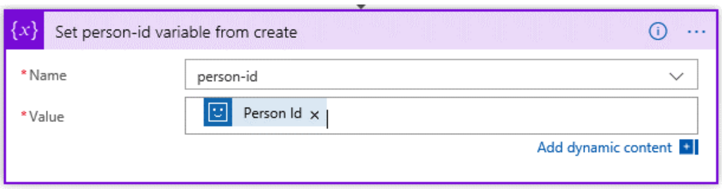
* Person Group Id: person-group-id ×

* Name: person-name ×

Connected to face-api. Change connection.

Set the person-id Variable if one was Created

- Click the **+** below the **Create a person** action and select **Add an action**, search for **Set Variable** and select the **Variables - Set Variable** action
- Click the **Name** field and select **person-id**
- Click the **Value** field and select **Person Id** under Create a person from the popup
- Click on the **Ellipsis (...) menu**, select **Rename** and name this Action **Set person-id variable from create**
- Your action should look like this



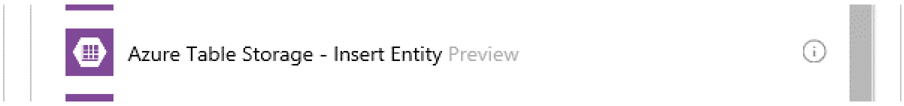
Set person-id variable from create

*Name: person-id

*Value: Person Id ×

Save the Person ID if one was Created

- Click the **+** below the **Set person-id variable from create** action and select **Add an action**, search for **Table Storage** and select the **Azure Table Storage - Insert Entity** action



Azure Table Storage - Insert Entity Preview

- Click the **Table** field and select **PersonIds** from the dropdown

- The **Entity** field is a JSON object containing the fields to be inserted into the entity, it must contain **PartitionKey** and **RowKey** properties and we will also add an **Id** field
- Copy the following empty JSON object into the **Entity** field

```
{
    "Id": "",
    "PartitionKey": "",
    "RowKey": ""
}
```

- Place the cursor between the empty quotes **""** after **"Id"** and click **person-id** from the popup; repeat this for **PartitionKey** and **person-group-id**, and **RowKey** and **person-name**
- Your action should look like this

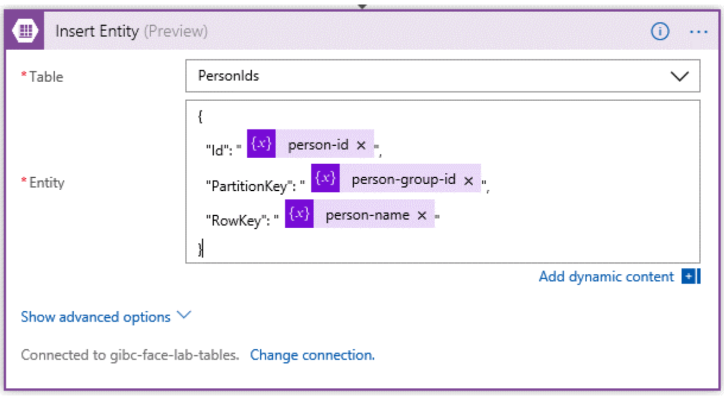


*Table: PersonIds

*Entity:

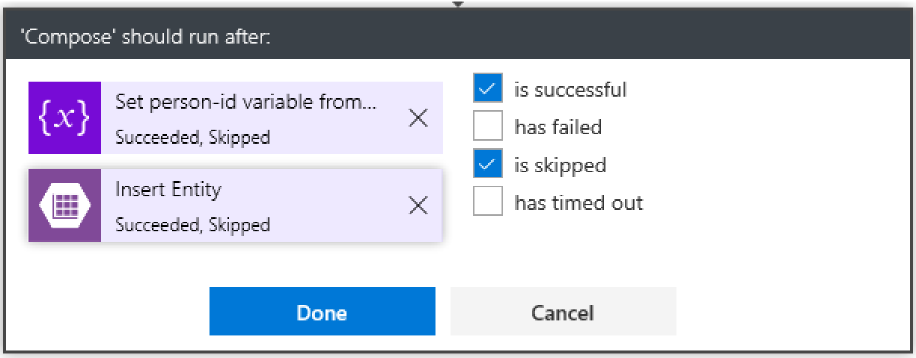
```
{
  "Id": "person-id",
  "PartitionKey": "person-group-id",
  "RowKey": "person-name"
}
```

Connected to gibc-face-lab-tables. Change connection.

Joining the Branches

- Note that one of the branches we've created will always be skipped, and that means that the final state of the scope will be set to *skipped* unless we do something about it
- To fix this we'll join the branches together with an empty action which we'll set to succeed if *either* of the previous branches is successful
- At the bottom of the **scope** click the **Add an action** button, search for **Compose** and select the **Data Operations – Compose** action

- Click in the **Inputs** field and set the contents to an empty JSON object
  { }
- Click the **Ellipsis (...) menu** and select **Configure run after**
- Click on the **Set person-id variable from...** tile and make sure **is successful** and **is skipped** are both ticked
- Click on the **Insert Entity** tile and make sure **is successful** and **is skipped** are both ticked
- The run after settings should look like this for both tiles



'Compose' should run after:

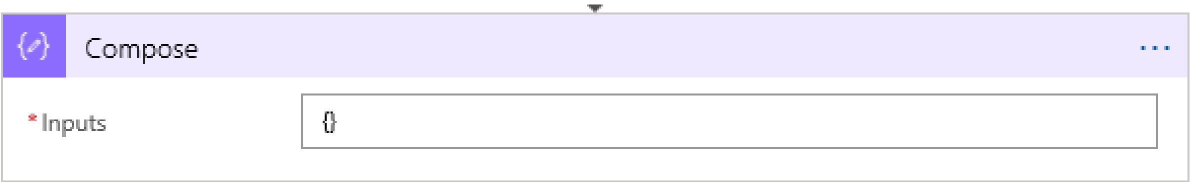
Set person-id variable from... — Succeeded, Skipped

Insert Entity — Succeeded, Skipped

☑ is successful
☐ has failed
☑ is skipped
☐ has timed out

- Click **Done**

- Your action should look like this

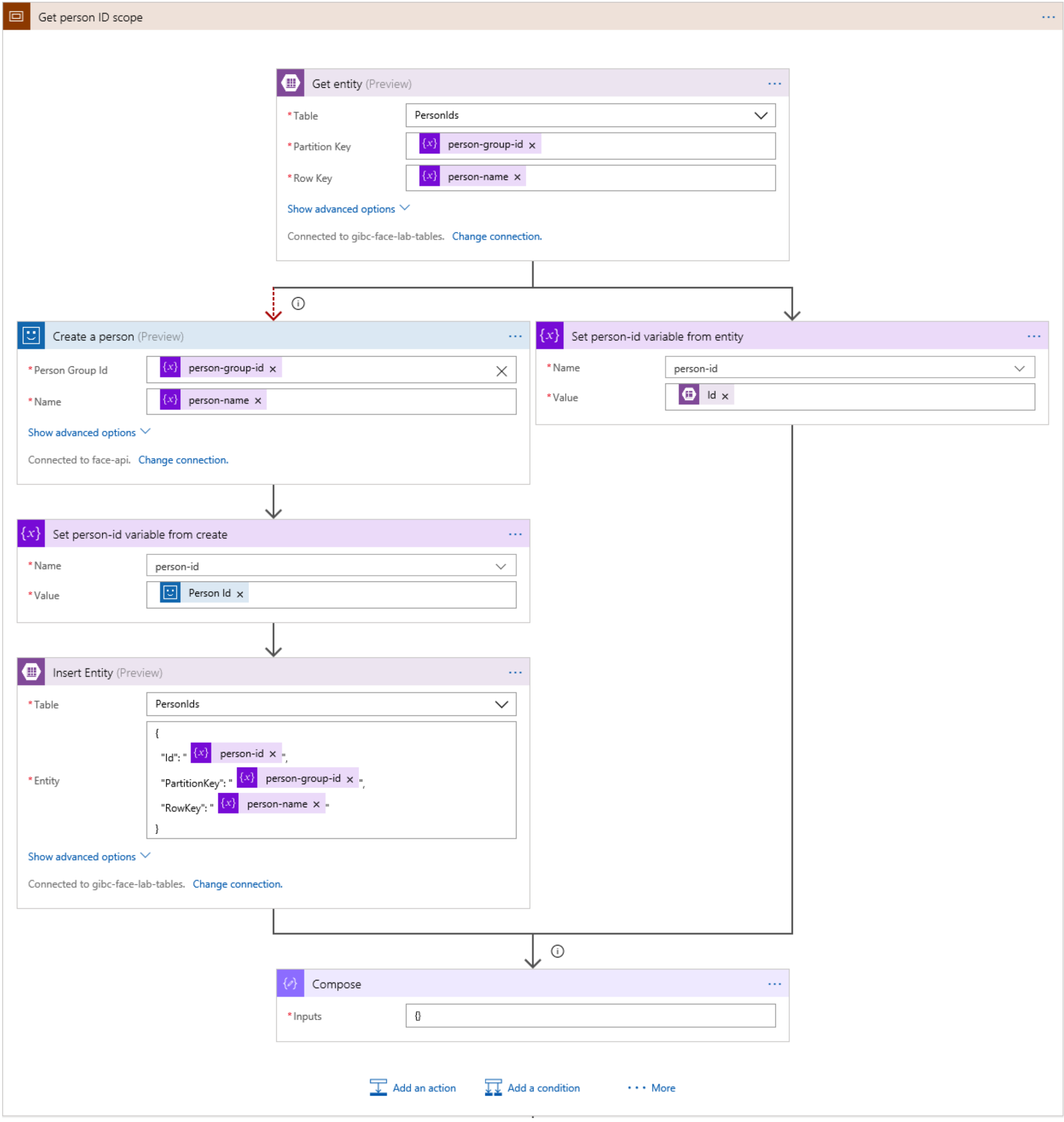


Compose

* Inputs: {}

What you Just Built

- This is the complete scope that you should have just created



Get person ID scope

Get entity (Preview)
- * Table: PersonIds
- * Partition Key: person-group-id
- * Row Key: person-name
- Show advanced options
- Connected to gibc-face-lab-tables. Change connection.

Create a person (Preview)
- * Person Group Id: person-group-id
- * Name: person-name
- Show advanced options
- Connected to face-api. Change connection.

Set person-id variable from entity
- * Name: person-id
- * Value: Id

Set person-id variable from create
- * Name: person-id
- * Value: Person Id

Insert Entity (Preview)
- * Table: PersonIds
- * Entity:

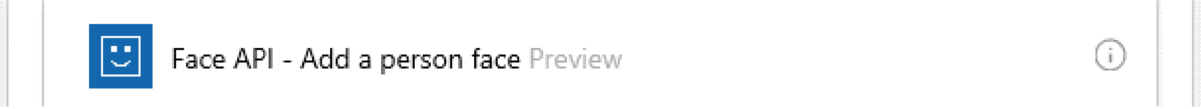
```
{
  "Id": " person-id ",
  "PartitionKey": " person-group-id ",
  "RowKey": " person-name "
}
```

- Show advanced options
- Connected to gibc-face-lab-tables. Change connection.

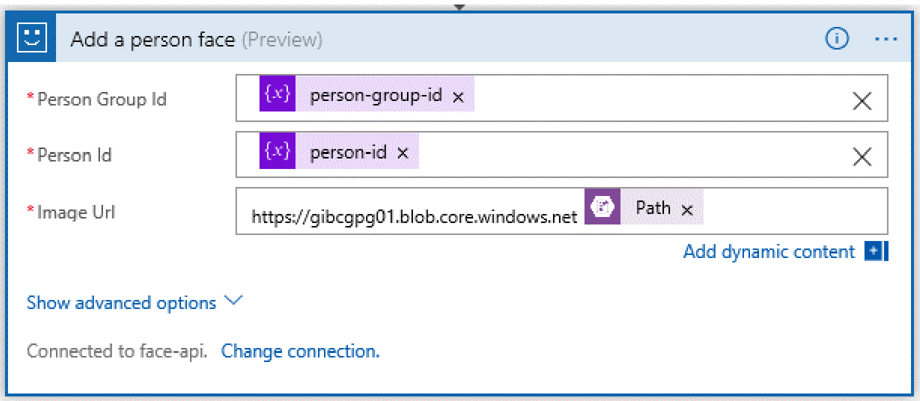
Compose
- * Inputs: {}

Add the Face Image to the Person

- In the Logic App click **+ New step**, search for **Add Person Face** and select the **Face API - Add a person face** action



- Click the **Person Group Id** field, click **Enter a custom value** and select the **person-group-id** Variable from the popup
- Click the **Person Id** field, click **Enter a custom value** and select the **person-id** Variable from the popup
- Click in the **Image Url** field. Enter the Blob endpoint you saved earlier without a trailing slash, e.g. https://gibcgpg01.blob.core.windows.net, and then with the cursor at the end of the URL click **Path** from the popup
- Your action should look like this



*Person Group Id: person-group-id

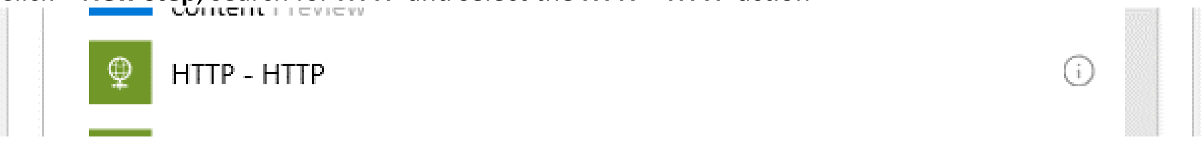
*Person Id: person-id

*Image Url: https://gibcgpg01.blob.core.windows.net Path

Connected to face-api. Change connection.

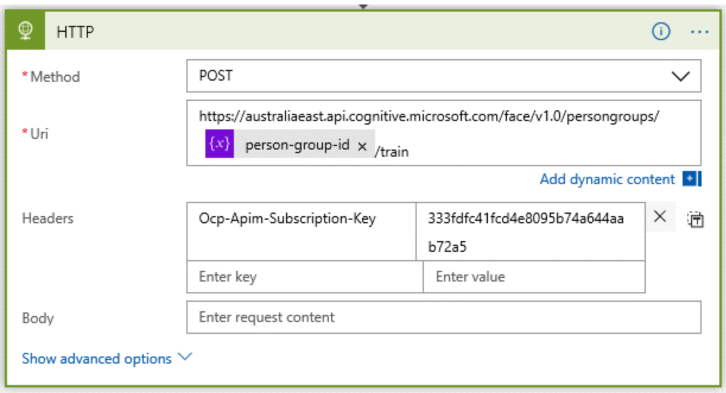
Training the model

- *Once faces have been added, removed or changed in a Person Group we need to train the model to incorporate those changes. We train the model on an entire Person Group. For this lab we're going to train the model after each training image is added, but in the real world if we were going to make many changes we'd perform the modifications by themselves and then train the model one time after all the changes were completed. We might even train the model on a schedule, such as once a day, depending on our needs.*
- Click **+ New step**, search for **HTTP** and select the **HTTP - HTTP** action



- Set **Method** to **POST**
- Set **URL** to `https://australiaeast.api.cognitive.microsoft.com/face/v1.0/persongroups//train`
  Note the double slash before **train**, we're going to put a variable in between them.
- Place the cursor in between the **//** before **train**, and select the **person-group-id** variable from the popup
- Set the first header **key** to **Ocp-Apim-Subscription-Key** and the **value** to the **Face API Key** you saved when you created your Face API instance.

- Your action should look like this



HTTP

*Method: POST

*Uri: https://australiaeast.api.cognitive.microsoft.com/face/v1.0/persongroups/ {x} person-group-id × /train

Headers: Ocp-Apim-Subscription-Key | 333fdfc41fcd4e8095b74a644aab72a5

Body: Enter request content

Try it Out

- Drop one or more training photos named in this format [Person Name]-[Number].jpg (e.g. Amanda Citizen-01.jpg) into the training folder that you selected for the OneDrive trigger
- After a few seconds the image should be picked up, a person ID retrieved or created, and the person group trained
- You can use the Logic Apps **Overview** to see the status of runs
- You can now go back to your identify-faces Logic App and change the Face API key to your own key, and try recognising faces for people that you've trained a model on
- *NOTE: If you drop multiple image files into the folder at the same time you may see some runs marked as failed. This is because you can't start training the model while another training session is already running. In reality you would create a mechanism so that this didn't occur, but for the purposes of simplicity this has been omitted for this lab.*