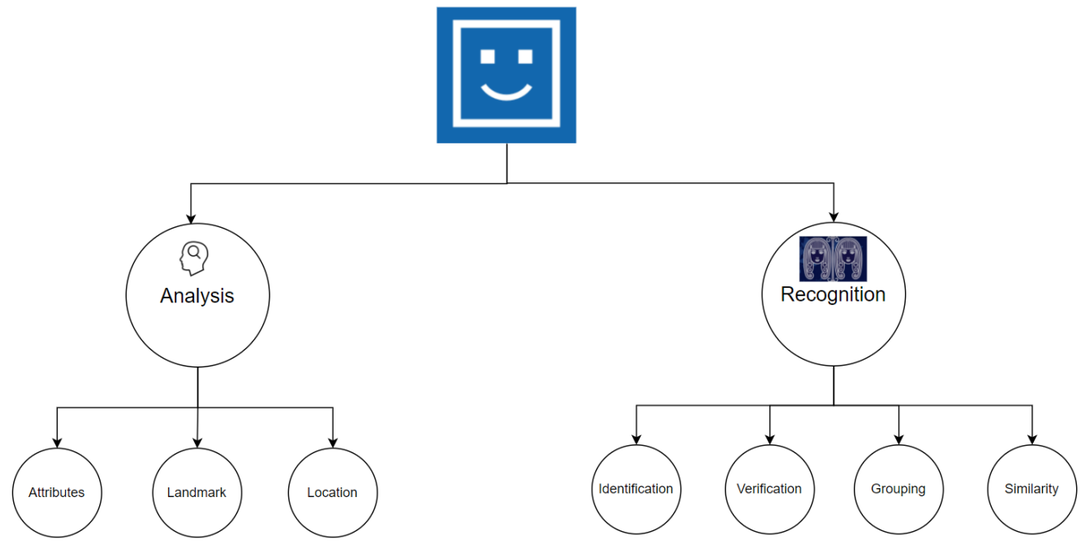
| | |
|---|---|
| **Document Name** | HOL – Azure Face Service v1.0 |
| **Author** | Shiva S Tomar & Anupreet Kaur |
| **Reviewer** | |
| **Executive Summary** | Azure Cognitive APIs enable the developers of all skill levels to add human intelligence in their applications. The services are designed for developers interested in pursuing DS/AI/ML skills and people who want to acquire the deep technical knowledge on the Cognitive APIs of Azure, despite not having Machine Learning expertise. |
| **Purpose** | This document is created to help you gain level 350 working knowledge on Azure Face Service. You will be able to explore each functionality offered by the service through the API and observe the outcomes. We have also shared a sample dataset to replicate what we have used to create the content of this workshop.<br>Once you complete these labs, you'll go from ***Zero to Hero*** on the respective Azure Cognitive service and should be able to ***D****emo, **D**evelop and **D**eploy* your own custom use cases. The important thing to note here is that you don't need to refer any other documents to complete this workshop. |
| **Intent of Guide** | This workshop is designed to help you explore all the features of a service offered through their APIs. The diagram shown in the beginning of the document is its functional Architecture; talking about the functionalities offered by the service in a flow. It also covers the Concepts, How-to and best practices about the service.<br>This document is not intended to enable you with scenarios of deployment in production. |

Service brief: Azure Face Service

Azure Face Service allows you to leverage advanced face recognition algorithms that help you detect & analyse human faces in an image, group similar faces or verify and identify people. You can incorporate this into your applications belonging to any business domain, where there is a need for face detection & analyses or face recognition.

Diagram: Functional Architecture




Azure Face Service functionality can be majorly divided into 2 buckets –

1. Face Detection & Analyses
2. Face Recognition

Let's now dive into each of them in a bit more detail.

Face Detection & Analysis

This allows you to detect all faces present in an image and return other face related details that help you to further analyse the face, such as –

1. Face Attributes : Fetches you details about the person & the face, such as age, emotions, gender and whether the person is wearing a mask, glasses or makeup. Furthermore, let's you know the face occlusion, blur, head pose, noise etc. You can use these for building use cases like liveliness detection etc
2. Face Landmarks : Fetches you the position of important face components such as eyes, pupils, nose, lips, mouth.
3. Face location : Fetches you the coordinates of the face is in the image. You can use this for cropping, face blurring etc

Face detection also returns a unique FaceID for every face detected in the image. You will leverage this when working with Face Verification & Identification. More on this later!

Face Recognition

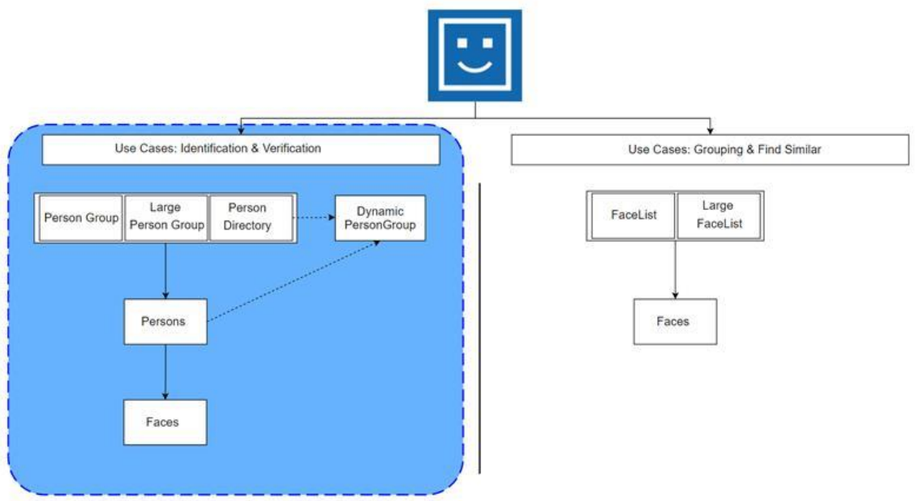
Like we saw in the functional architecture diagram above, Face Recognition feature has 4 functionalities in it -

1. Identification : refers to finding whom does the input face match to out of a collection of people. This is like a one-to-many matching.

2. Verification :  refers to finding whether the 2 input faces belong to the same person. This is like a one-to-one matching.
3. Grouping : refers to dividing a collection of input faces into groups based on face similarity. Each group may or may not contain faces belonging to the same person.
4. Find Similar : refers to finding similar looking faces to a given input face.

Let's now talk about how you can get started with **Face Identification & Verification**. For this, you will have to first build a collection of people & their corresponding faces, to train the model for your use case.

The hierarchy of objects you will have to create looks something like the blue box on the left, in the image below.




Does this look confusing? Let's understand what they really mean.

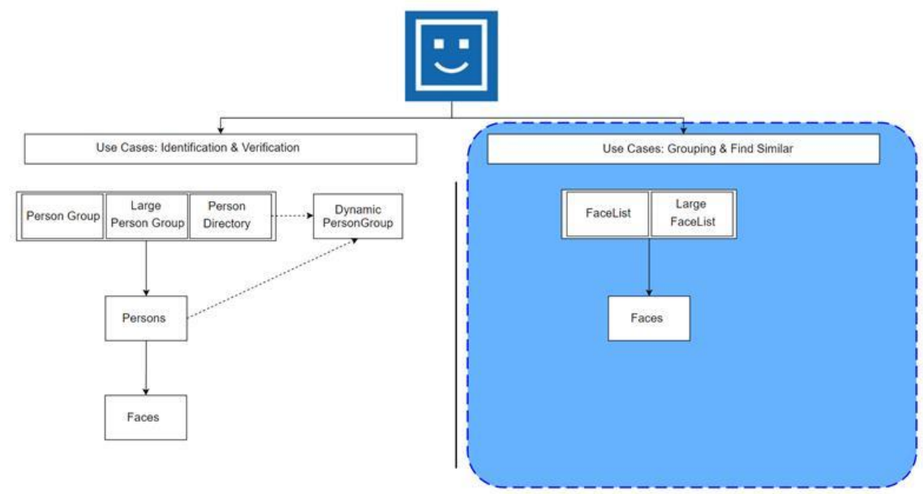
**Concepts & Terminologies**

** Wherever in the workshop we have used 'Person Object', it simply means an individual. Similarly, a 'Face Object' means 1 facial image of an individual.

1. Face : This represents a single face that is stored as part of your model creation. Each face is associated with a unique persistedFaceId, which is returned by the API when you add a face object in your model collection.
   The image of the person is not stored by the service, only the extracted features are stored. You cannot reverse engineer a face using these features! Responsible AI, eh?
2. Person : This represents a collection of faces belonging to the same person. It has a unique ID, a name string, and optionally user data that can be used for meta data information. Each Person object can hold a maximum of 248 Face objects.
3. Person Group : This represents a collection of Person objects. You can create a maximum of 1 Mn Person Groups and each Person Group can hold up to 10,000 Person objects.
4. Large Person Group : Like Person Group, this also represents a collection of Person objects. How does this differ from Person Group then, you ask?
   The only difference lies in the quota limitations. You can create a maximum of 1 Mn Large Person Groups and each Large Person Group can hold up to 1 Mn Person objects.
5. Person Directory : This also represents a collection of Person objects. This further enhances the limit of managing Person objects up to 75 Mn under 1 Person Directory. Each service has 1 default Person Directory associated with it. You cannot create additional Person Directory. This is the latest addition to the Face Service.
6. Dynamic Person Group : This represents a subset of the Person Directory and contains a collection of references to Person objects within a Person Directory. Hence it is essential to first add Person objects to the default Person Directory before creating Dynamic Person Groups.

Let's now talk about how you can get started with **Grouping & Find Similar** functionalities. Just like Identification & Verification, you will have to first build a collection of Face Lists & Faces in the Face List, to train the model for your use case.

The hierarchy of objects you will have to create looks something like the blue box on the right, in the image below.



Let's understand what they really mean.

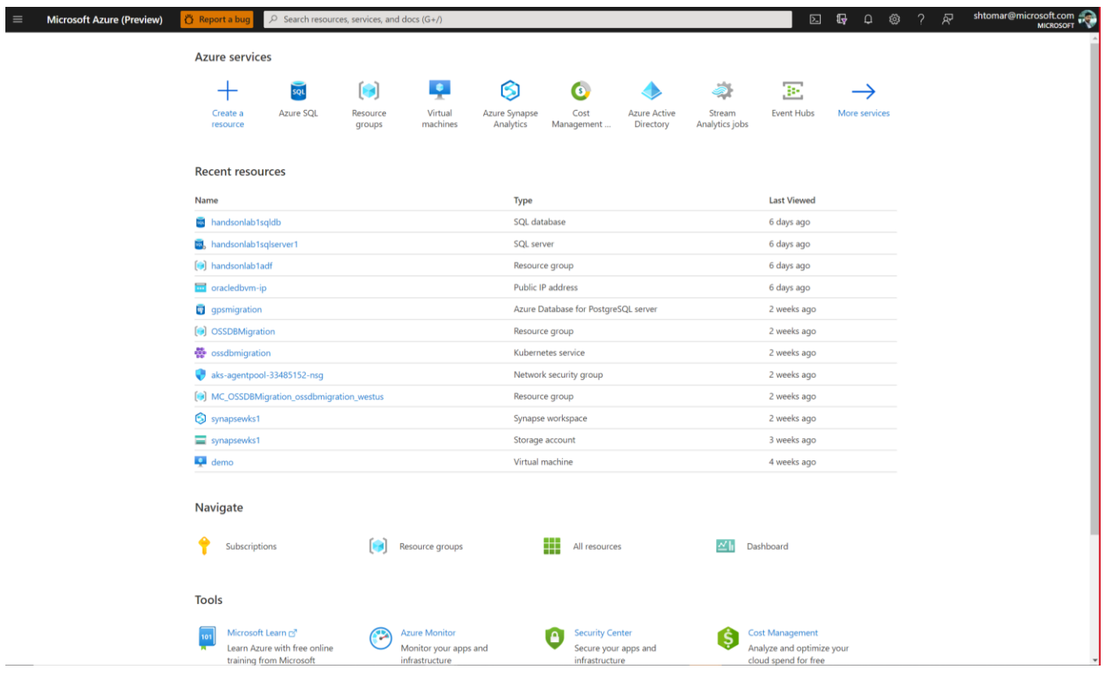
1. **Face** : This represents a single face that is stored as part of your model creation. Each face is associated with a unique persistedFaceId, which is returned by the API when you add a face object in your model collection. This is the same as the Face object in Verification & Identification object hierarchy.
2. **FaceList** : This represents a collection of faces. These may or may not belong to the same person. It has a unique ID, a name string, and optionally user data that can be used for meta data information. You can create a maximum of 64 FaceLists per service and each FaceList can hold up to 1,000 Face objects.
3. **Large FaceList** : Like a FaceList, this also represents a collection of faces. The only difference is that you can create 1 Mn Large FaceLists per service and each Large FaceList can hold up to 1 Mn Face objects.

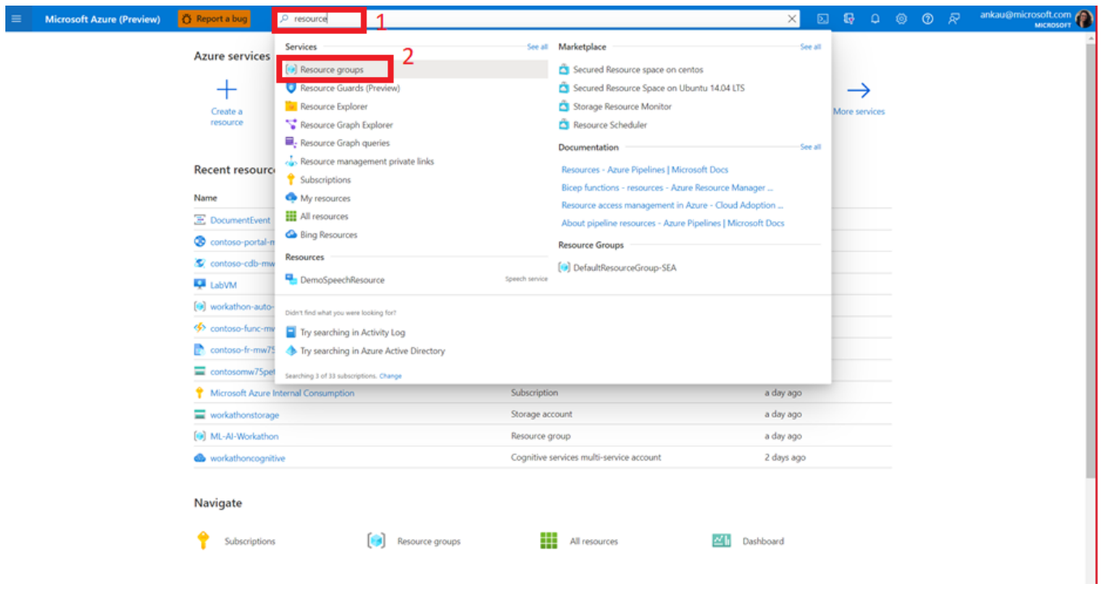
## Step by step hands on guide to go from *Zero to Hero*

### Pre-requisites

- Download & Install Postman
  - Postman is a free tool which allows you to make API calls
  - You can download the desktop application or get started using the web version (Download Postman | Try Postman for Free)
- An active Azure Account
  - You can use your current Azure Subscription or get started by creating a free trial account (https://azure.microsoft.com/en-in/free)
- Download the data for training & testing from the Data folder in Face Service folder

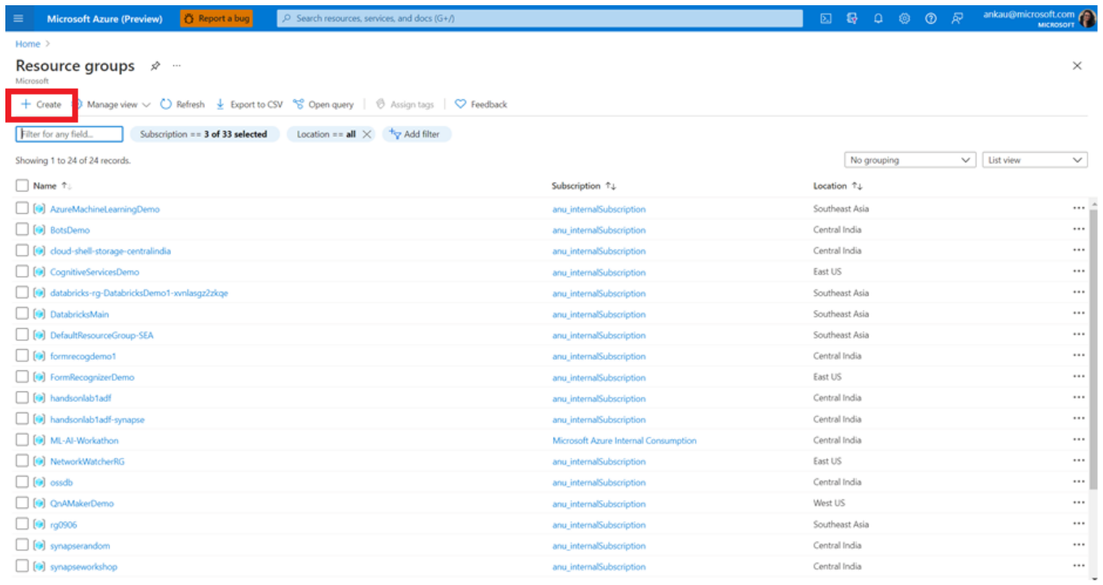
Let's get started!

| Screenshots | Steps & Significance |
| --- | --- |
|  | Sign into your Azure Portal. |

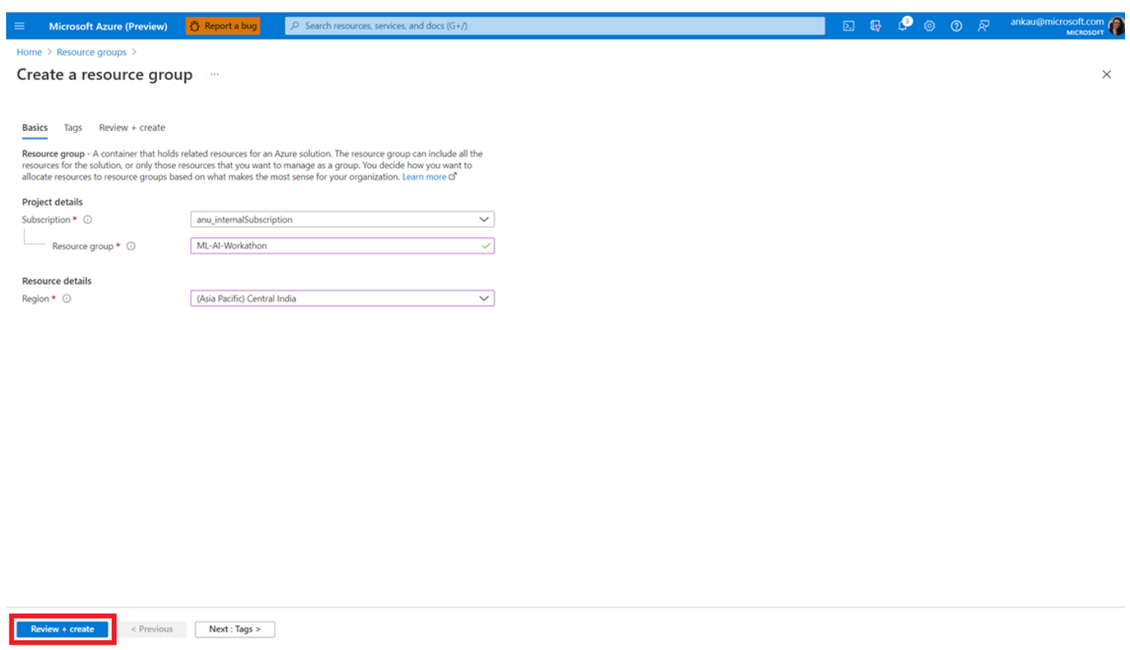



## Create a Resource Group

Follow steps 1 & 2 to create a resource group.



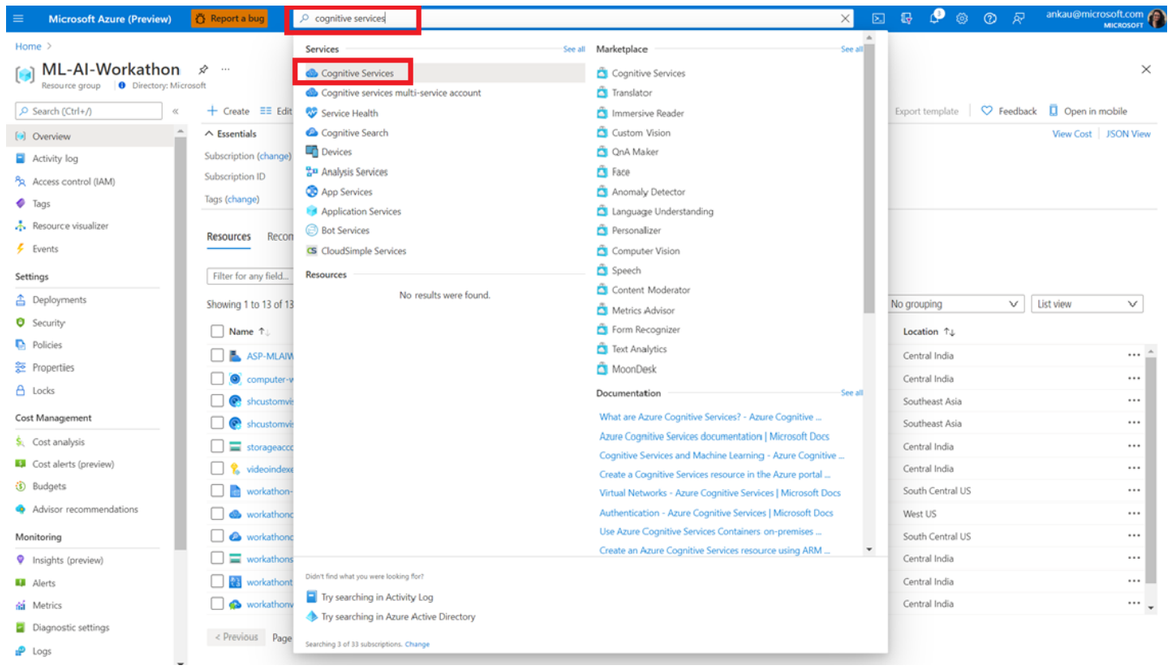

Click create to create a new resource group.



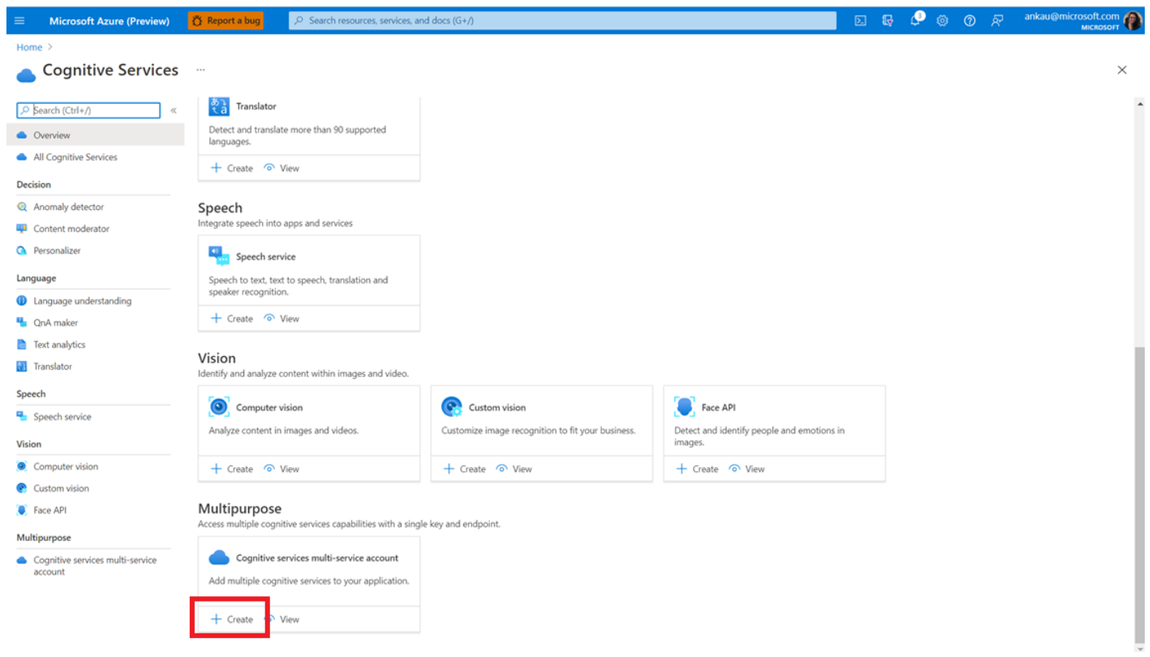

Enter the details –

1. Subscription : Azure subscription in which you want to deploy the resource group
2. Resource Group : Name of your choice for the resource group
3. Region : Region where you want to deploy the resource group

Click Review + Create.




Once the resource group is created, search for Cognitive Services in the search bar above and select Cognitive Services.

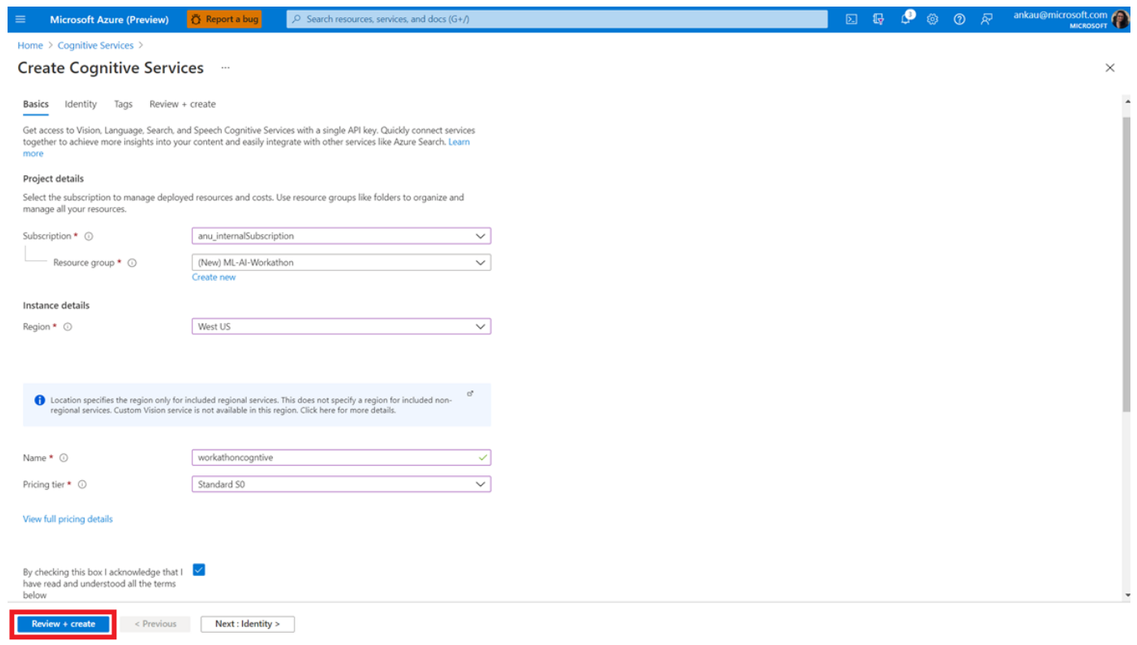



## Create a multipurpose cognitive service

Significance : A multipurpose Cognitive Service account allows you to leverage the same resource for many cognitive services, which include :

Computer Vision - Analyze images
Content Moderator - Check text, image or videos for offensive or undesirable content
Face - Recognize people and their attributes in an image
Form Recognizer - Identify and extract text, key/value pairs and table data from form documents
Language Understanding - Extract meaning from natural language
Speech - Transform speech-to-text, text-to-speech and recognize speakers
Text Analytics - Detect sentiment, key phrases, entities and human language type in text

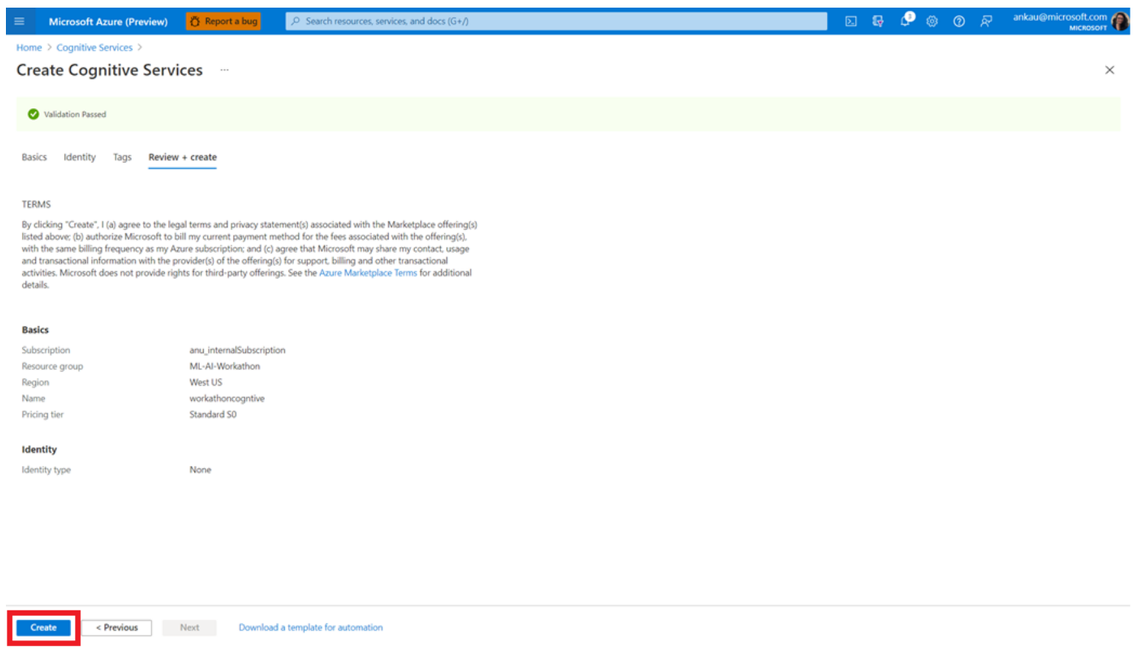
In this lab, we used a multipurpose Cognitive Service account since we would be learning about all the above-mentioned services.
However, you can also spin up individual services to execute these labs or for your development / production scenarios. The only difference is spinning up individual services allows logical separation from workspace standpoint and easy monitoring of billability.



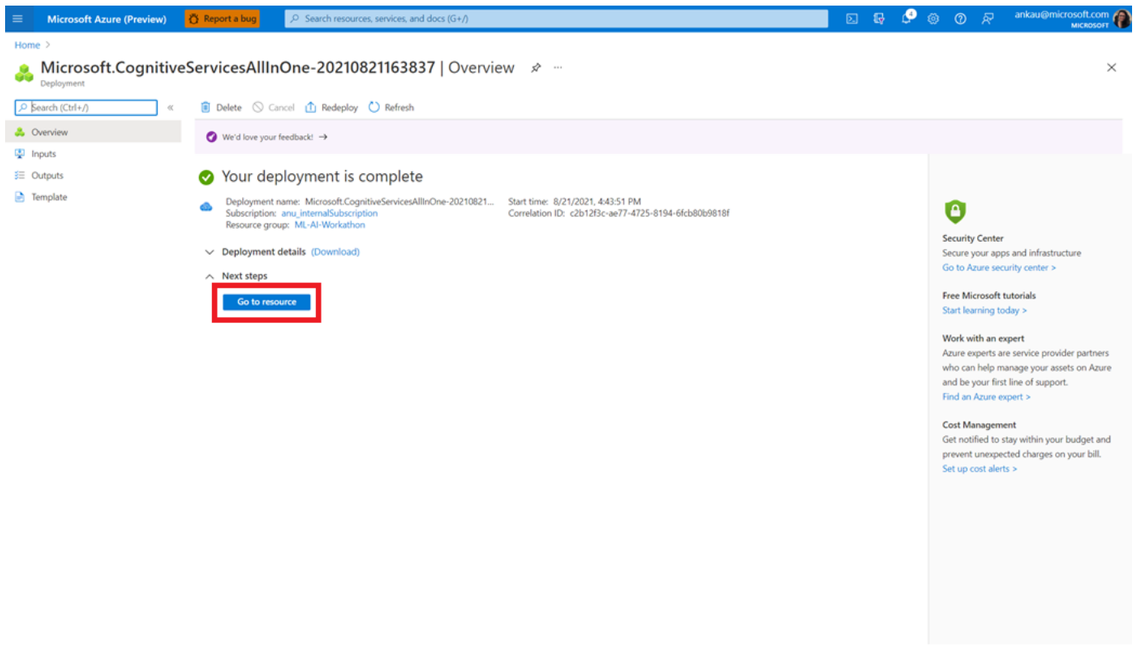

Enter the details to create a new cognitive service as follows -

| Project details | Description |
|---|---|
| Subscription | Select one of your available Azure subscriptions. |
| Resource group | The Azure resource group that will contain your Cognitive Services resource. You can create a new group or add it to a pre-existing group. |
| Region | The location of your cognitive service instance. Different locations may introduce latency but have no impact on the runtime availability of your resource. |
| Name | A descriptive name for your cognitive services resource. |
| Pricing tier | The cost of your Cognitive Services account depends on the options you choose and your usage. |

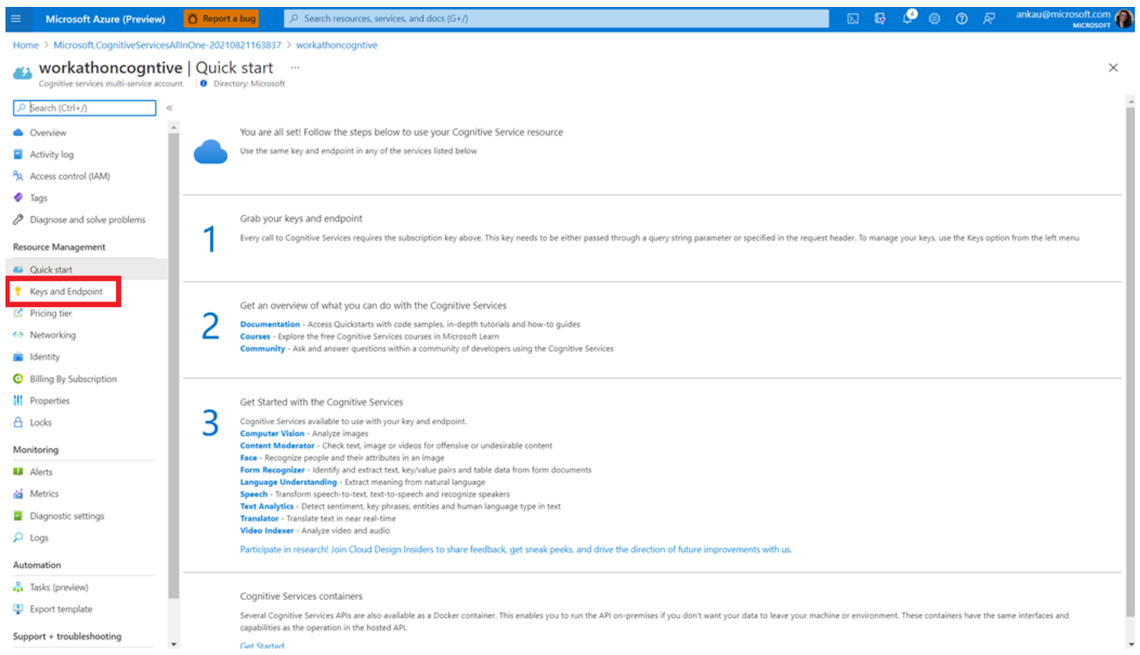
Click Review + Create.




Verify the details and click Create.



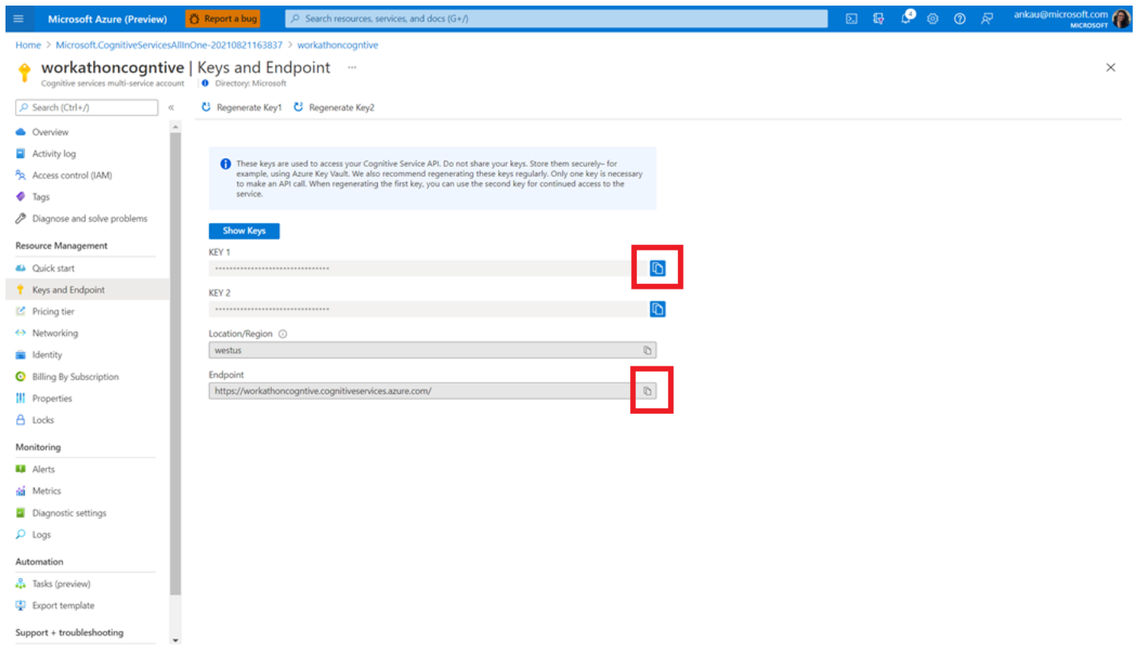

After the resource has been deployed, click Go to Resource.



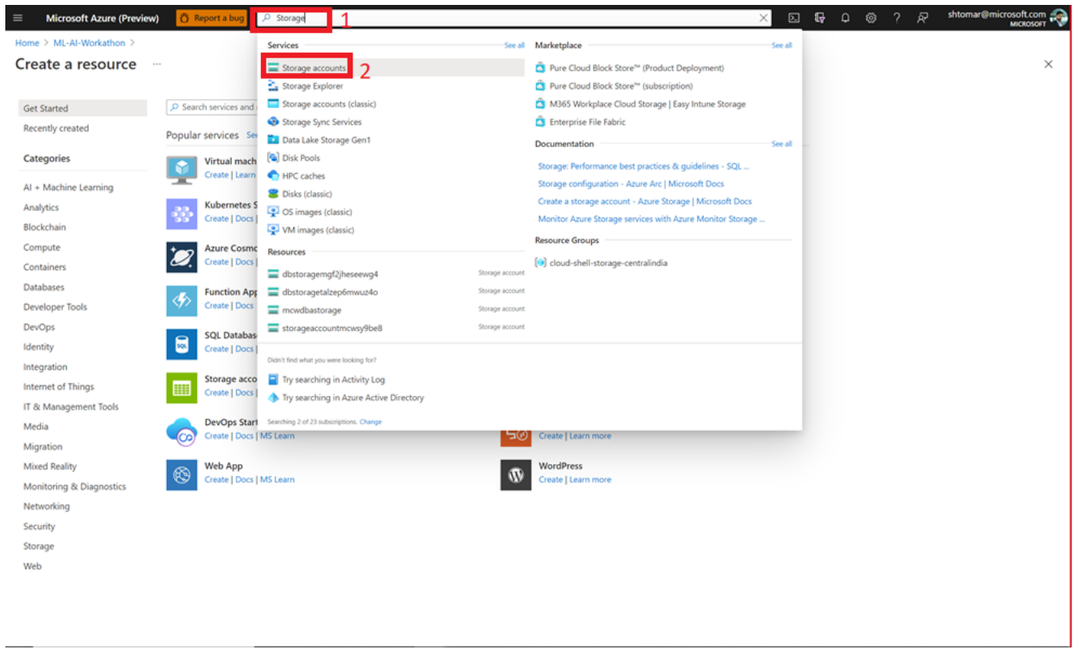

<u>Copy keys & endpoints</u>

On the Quick start page, you can find details about different cognitive services and can click the hyperlinks to learn more.

Click Key and Endpoints.



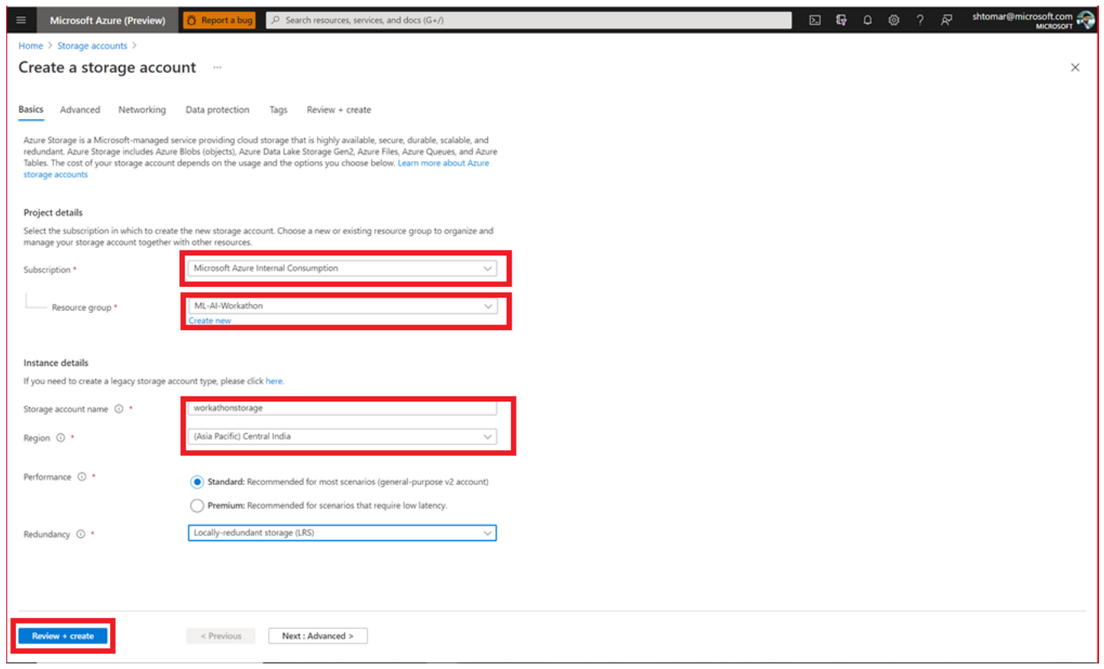
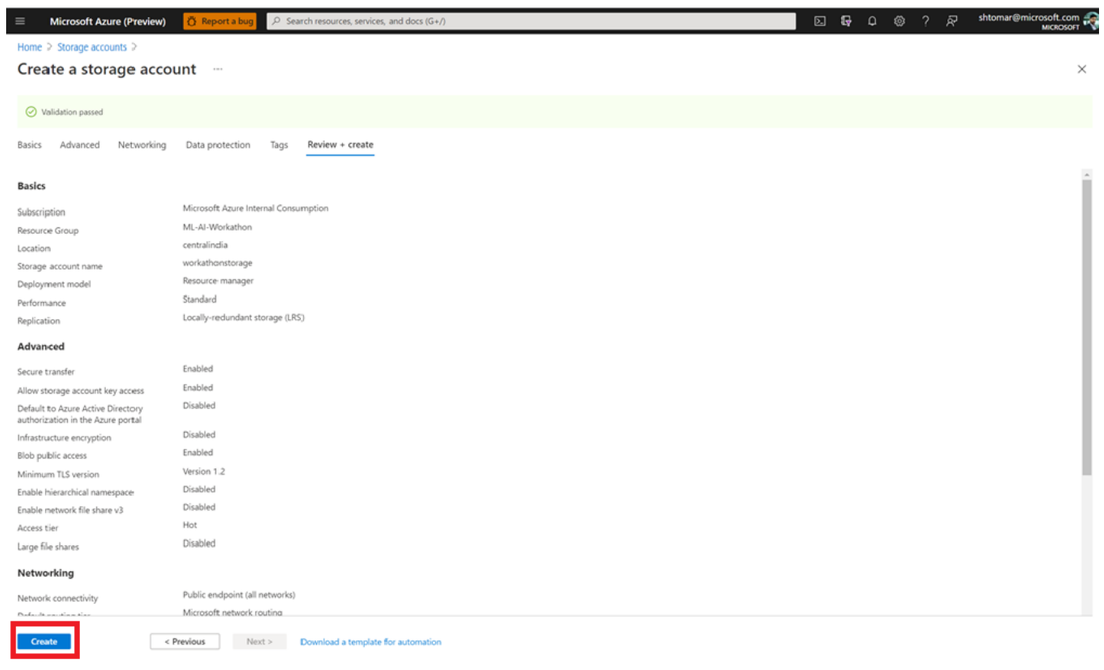

Copy the Key and Endpoint. Paste these in a notepad. You will leverage these at a later step, while setting up the global variables in Postman.

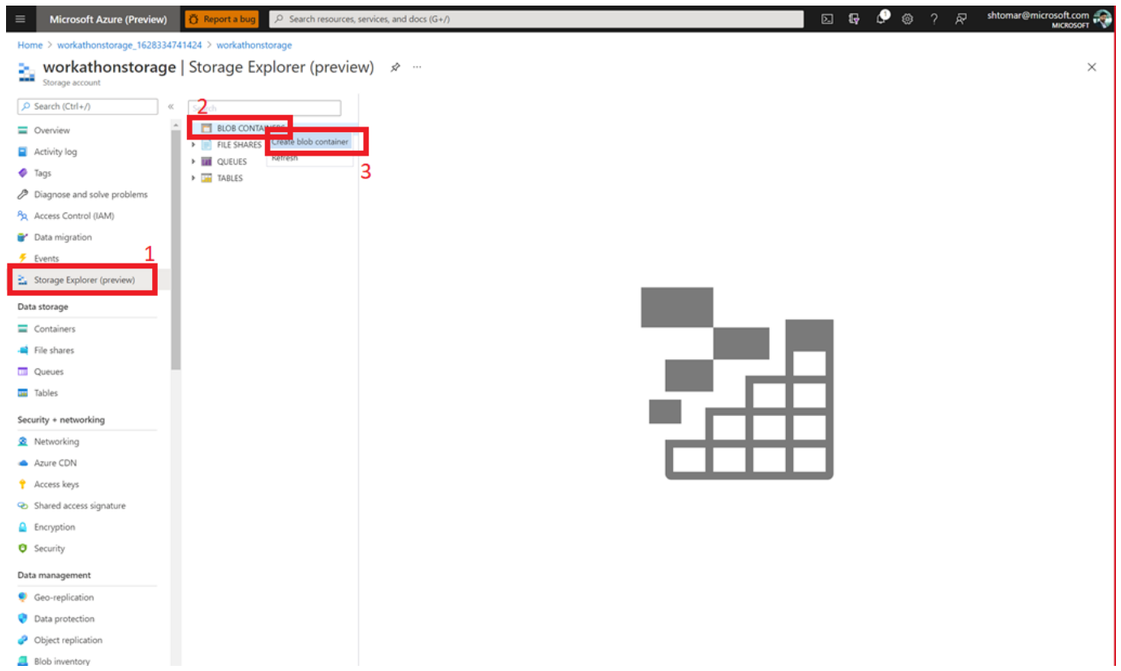



<u>Create Storage Account in Azure Portal</u>

This will be used to upload Face images used to train recognition model.
We will be uploading the training and testing images in the exact same hierarchy as the Data folder.

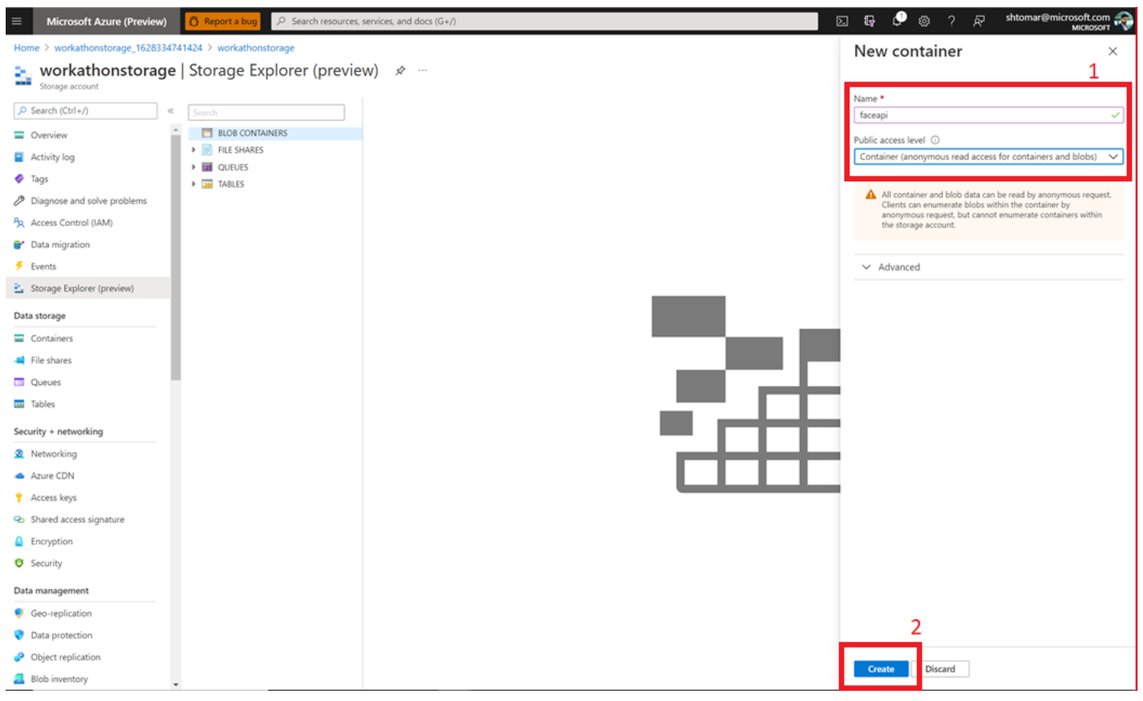
You can skip this step if you already have a Storage Account in place.

| | |
|---|---|
|  | Enter the details highlighted in the screenshot.<br>Click Review + Create. |
|  | Verify the details and hit Create. |




## Create Blob container

Go to the Storage Explorer (preview) to create a blob container.
Follows the steps as numbered in the screenshot.

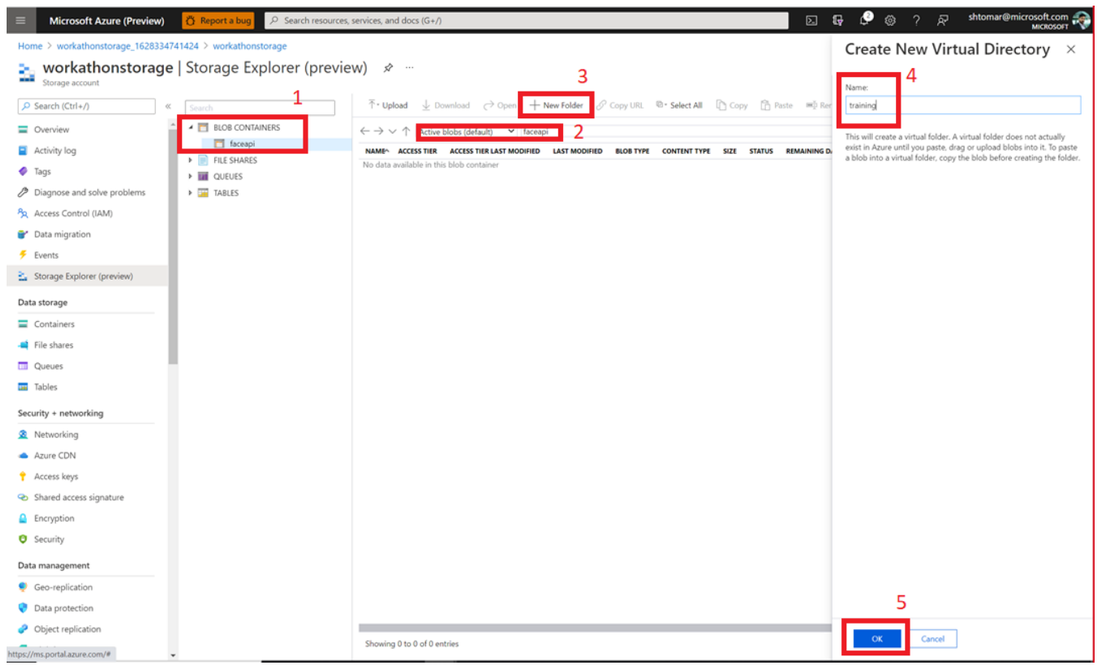



Create a container with the following details –

1. Name : **faceapi** [ You can choose a different name, however, that will change the path to your images while making API calls from postman. Make sure to change the image paths accordingly ]
2. Public access level : Container (anonymous read access… ) **

Click Create.

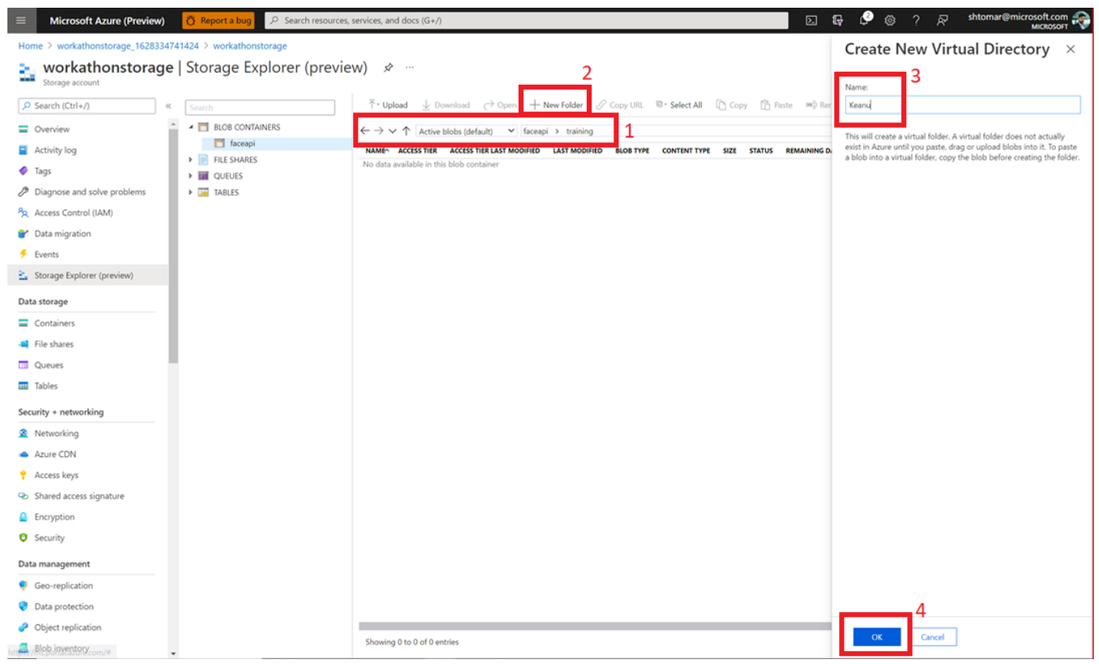
** This workshop is not considering security set up as this is designed to familiarise you with service capabilities.
We will be sharing the security best practices in a separate document.




<u>Create folder for training data</u>

Make sure you are in the **faceapi** container, as shown in steps 1,2.
Select + New Folder.
Name the folder – training. [ You can choose a different name, however, that will change the path to your images while making API calls from postman. Make sure to change the image paths accordingly ]
Click Ok.

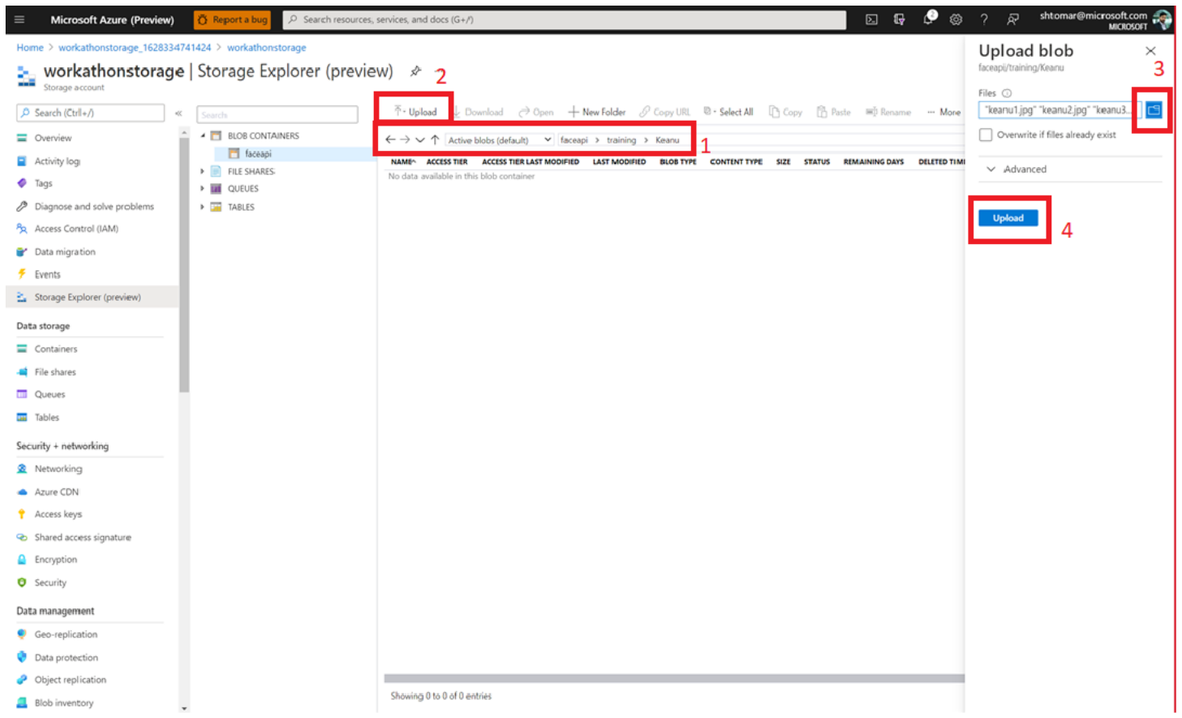
Once you click Ok, make sure to stay on the same screen and create another folder inside it.
As this is a virtual folder and will not be accessible if you leave it without adding any files. Once you add a file / folder, it will persist and you will be able to access the path.




<u>Create folder for Person 1 (Keanu)</u>

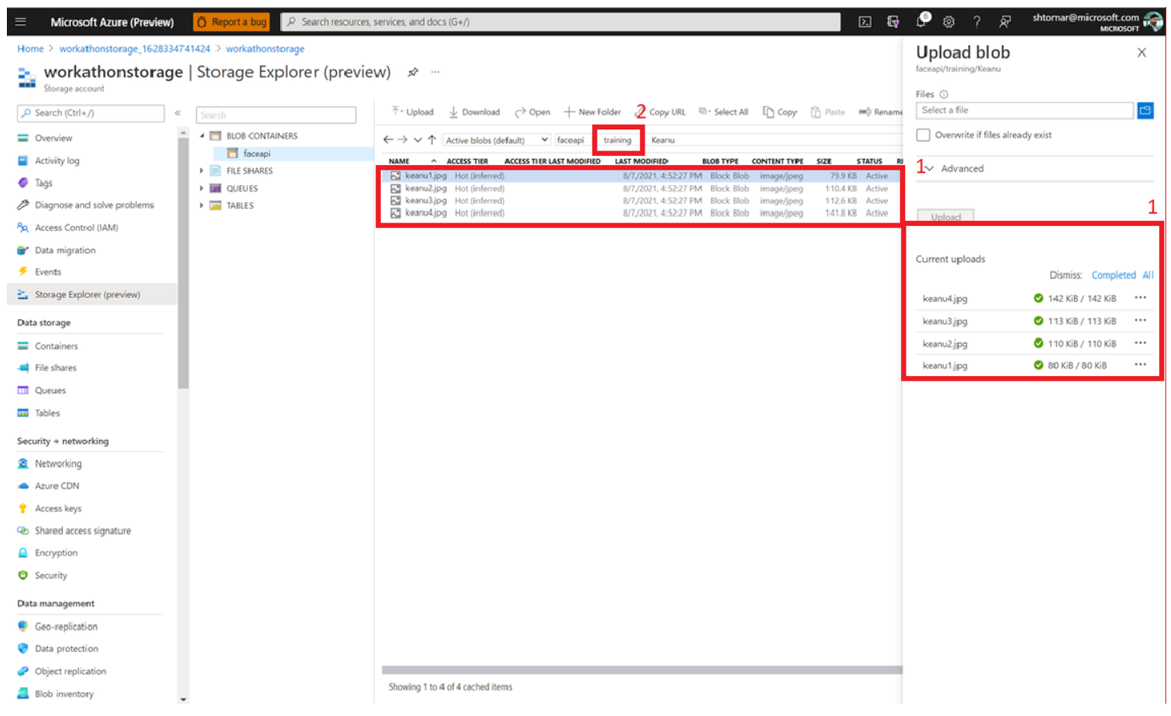
Make sure you are in the training folder, as shown in step 1.
Select + New Folder.
Name the folder – **Keanu**. [ You can choose a different name, however, that will change the path to your images while making API calls from postman. Make sure to change the image paths accordingly ]
Click Ok.

Make sure once you click Ok, make sure to stay on the same screen and create another folder inside it.
As this is a virtual folder and will not be accessible if you leave it without adding any files. Once you add a file / folder, it will persist and you will be able to access the path.




<u>Upload face images for person 1 (Keanu)</u>

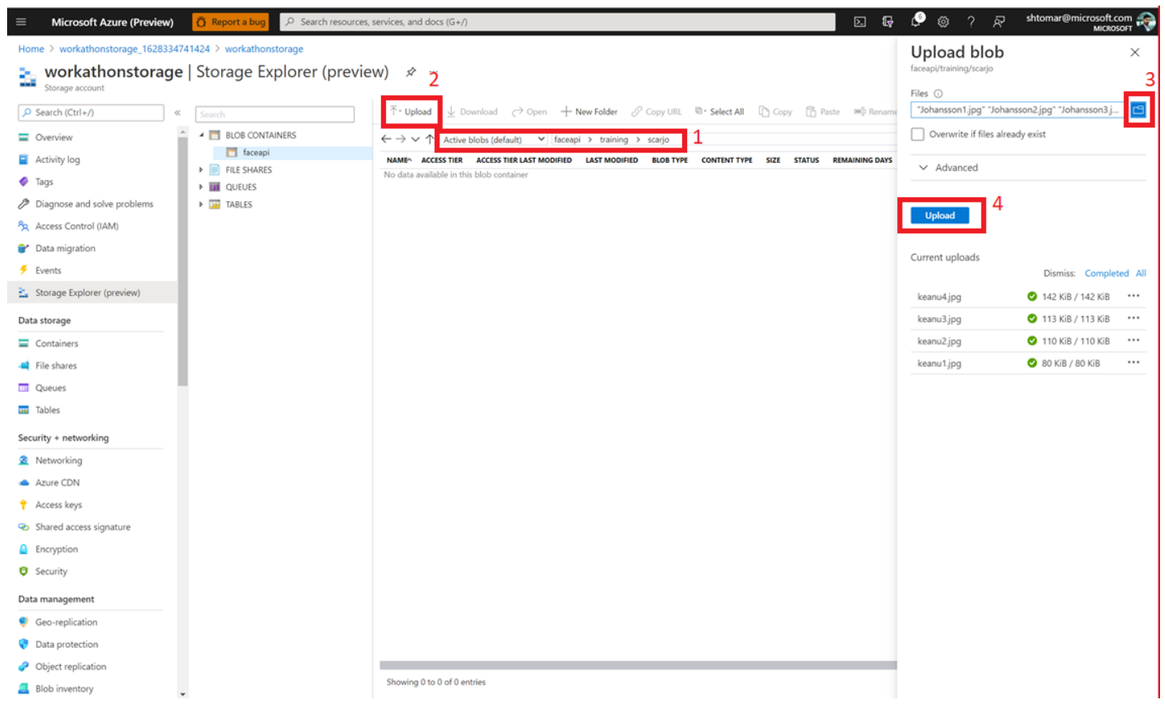
Make sure you are in the **Keanu** folder, as shown in step 1.
Select Upload as shown in step 2.
Click the file icon as shown in step 3 and browse & select the images you want to upload for person 1 [Keanu].
Click Upload button as shown in step 4.



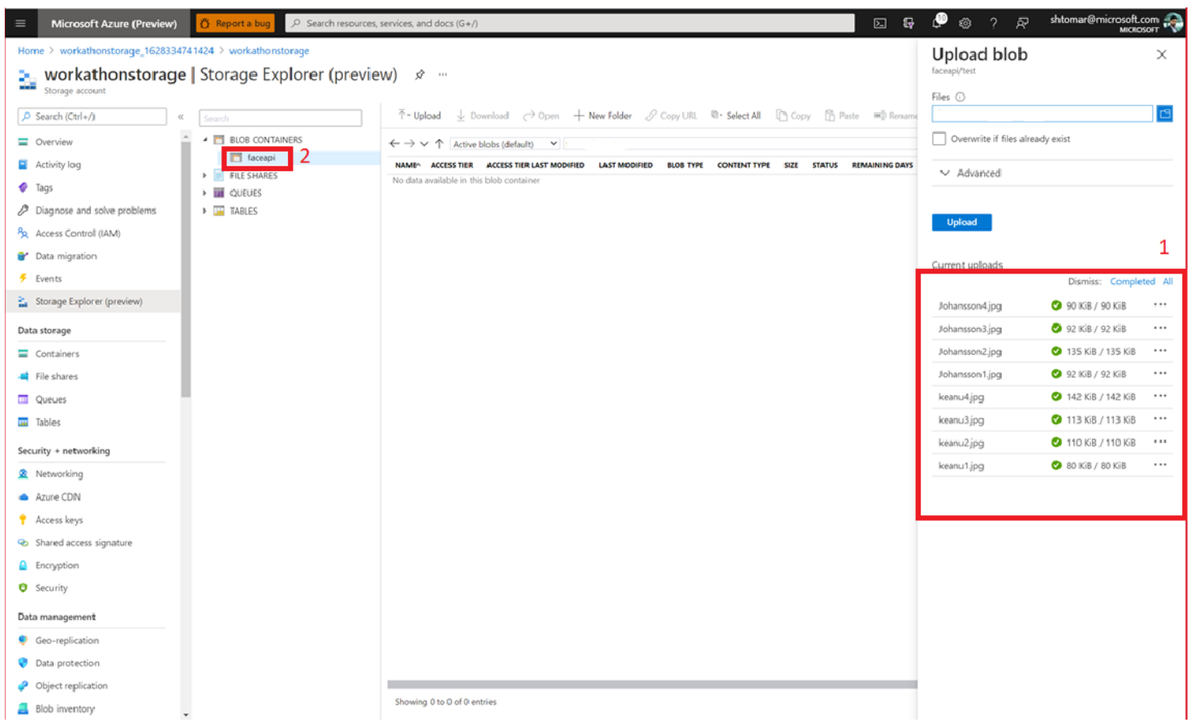

<u>Create folder & upload face images for Person 2 (scarjo)</u>

Observe the uploads as shown in step 1.

Now, select the training folder as shown in step 2.
Repeat steps performed in above 3 screenshots [where creation of Keanu folder started], to create a folder named '**scarjo'** and upload her face images.

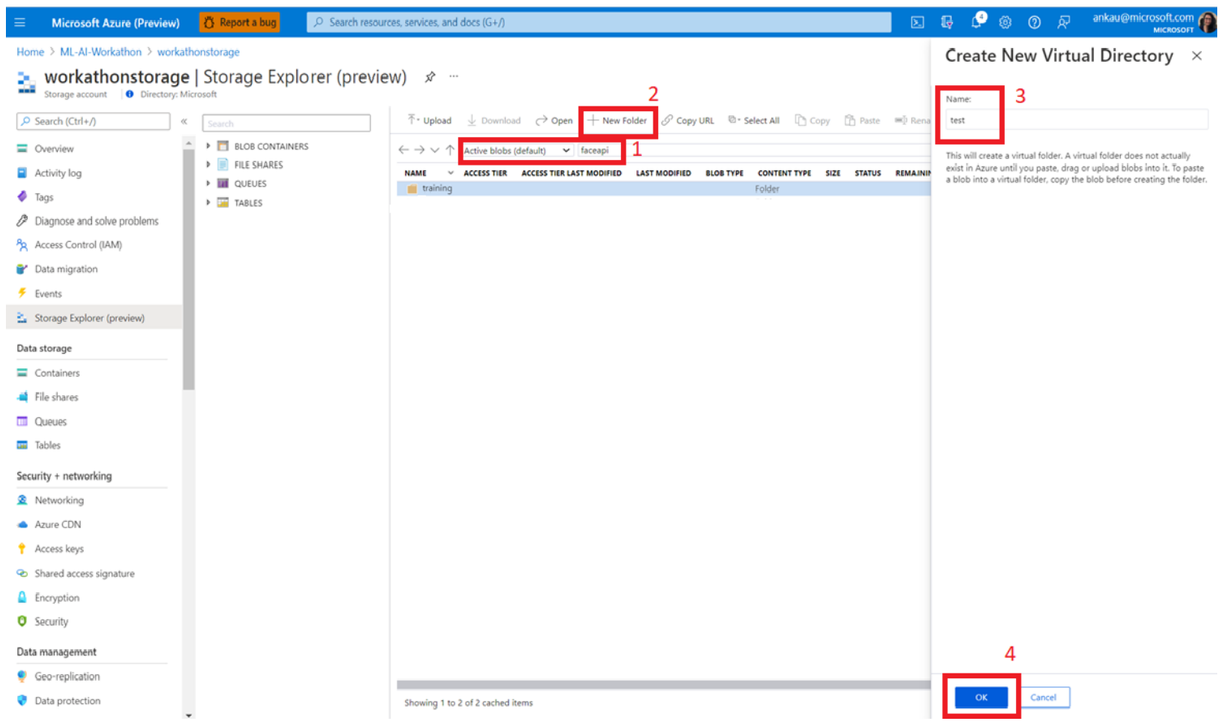



Once you create scarjo folder within training folder, select Upload as shown in step 2.
Click the file icon as shown in step 3 and browse & select the images you want to upload for person 2 [Scarjo].
Click Upload button as shown in step 4.




Observe the 8 pictures that have been uploaded in total, in step 1.
Click on faceapi container, as shown in step 2.

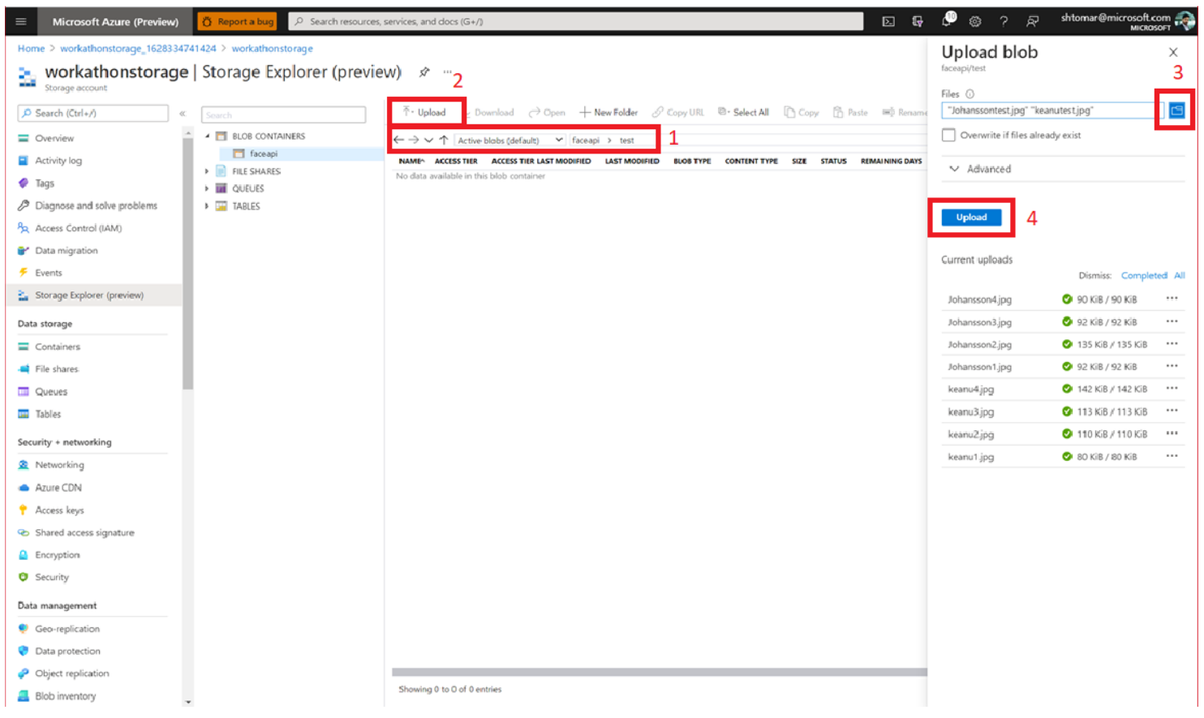
Like we created a training folder, we will now create a test folder and upload images to test the face recognition model we will be training.




<u>Create folder for testing data</u>

Make sure you are in the **faceapi** container, as shown in steps 1.
Select + New Folder.
Name the folder – **test**. [ You can choose a different name, however, that will change the path to your images while making API calls from postman. Make sure to change the image paths accordingly ]
Click Ok.

Make sure once you click Ok, make sure to stay on the same screen and create another folder inside it.
As this is a virtual folder and will not be accessible if you leave it without adding any files. Once you add a file / folder, it will persist and you will be able to access the path.



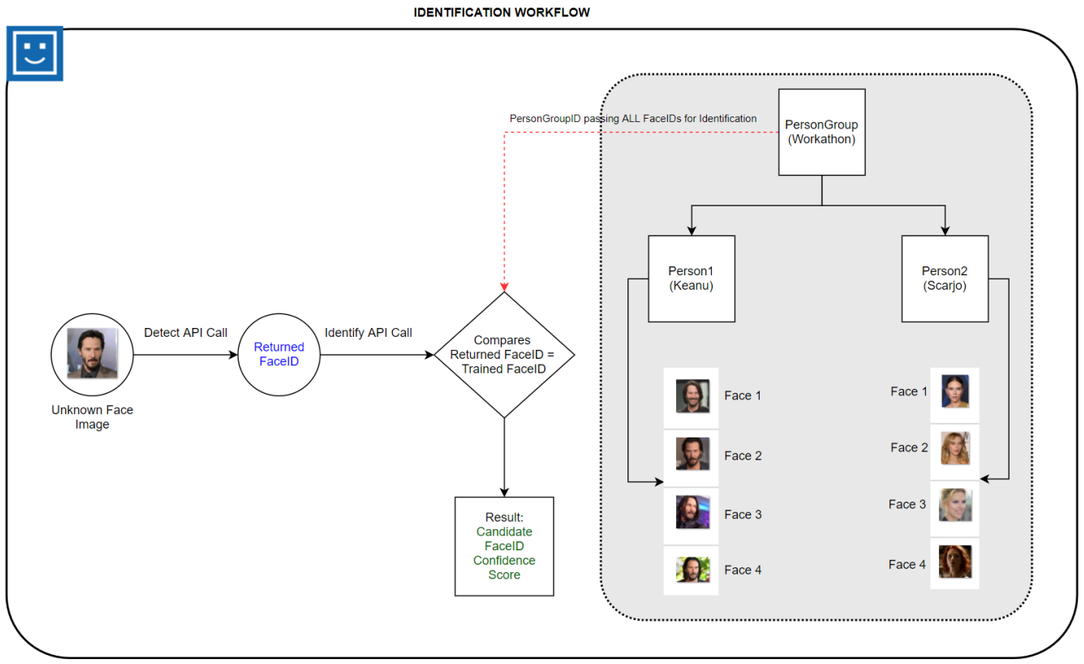

<u>Upload test images</u>

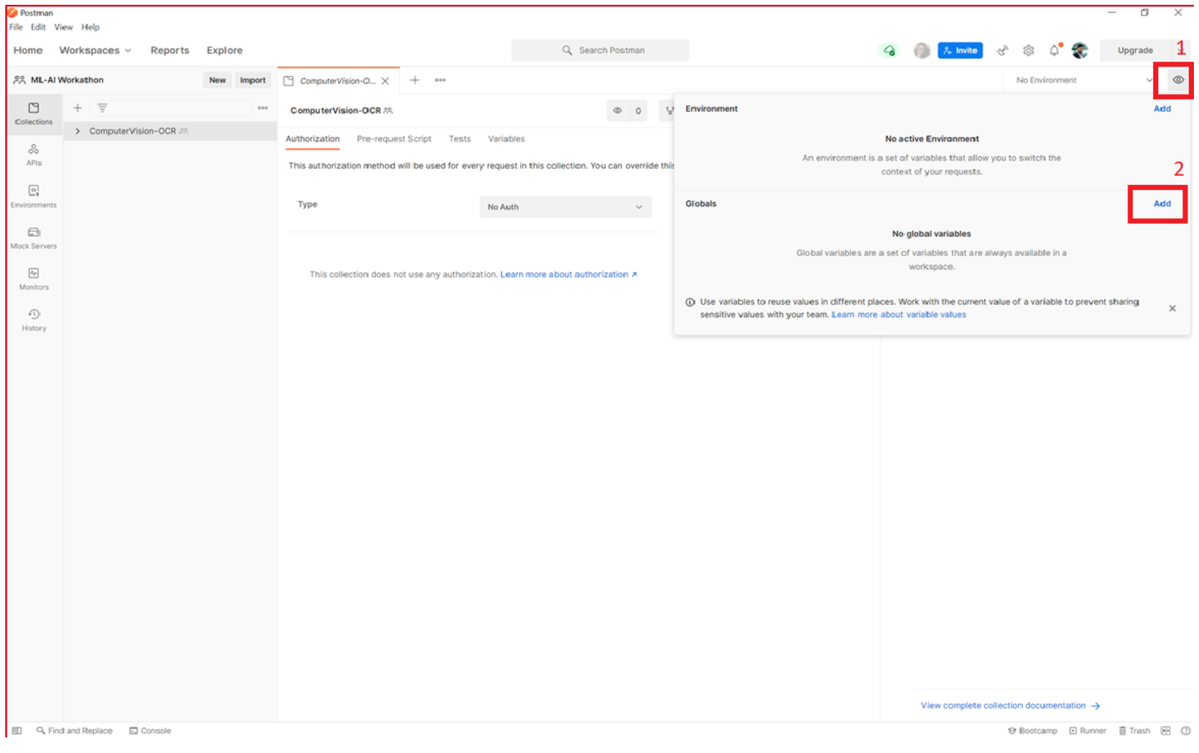
Make sure you are in the test folder, as shown in step 1.
Select Upload as shown in step 2.
Click the file icon as shown in step 3 and browse & select the images you want to upload for testing your model.
Click Upload button as shown in step 4.

## Identification use case in Azure Face Service

Now that we are ready with the necessary infra & data, let us explore the Face API service by executing the Identification workflow as shown in the diagram below.
We will first create the objects for model consumption which include Person Group, Persons, Faces – as shown in the grey area.
Once the objects are created, we will train the model, to execute the identification & verification use cases.
The objects shown in the grey area will also be leveraged by the verification use case, that we will implement after identification use case.

Note : In order to explore the Face Service and execute this workflow, we will be leveraging the Rest API calls fired through Postman.



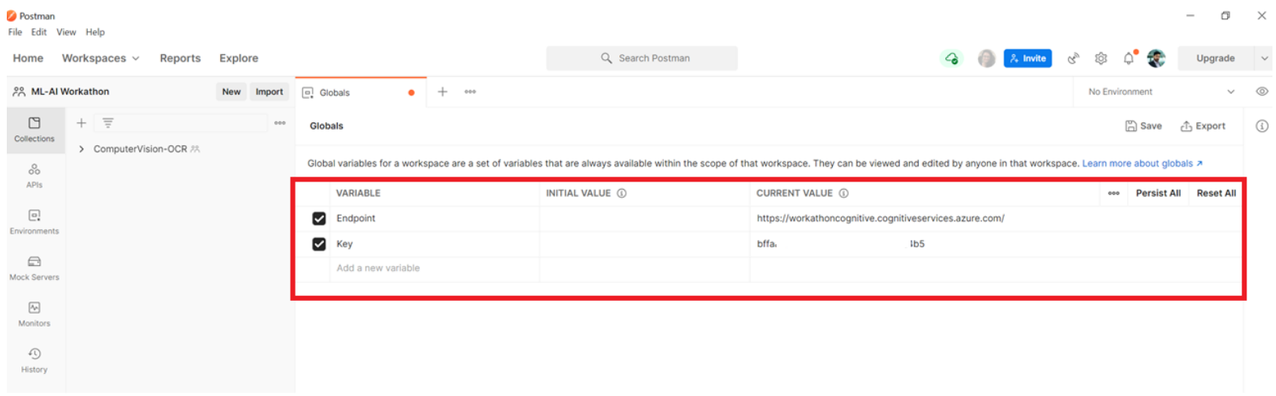

We have now switched the interface to Postman. If you haven't downloaded the Postman client, you can use web version.

### Configure global variables in Postman

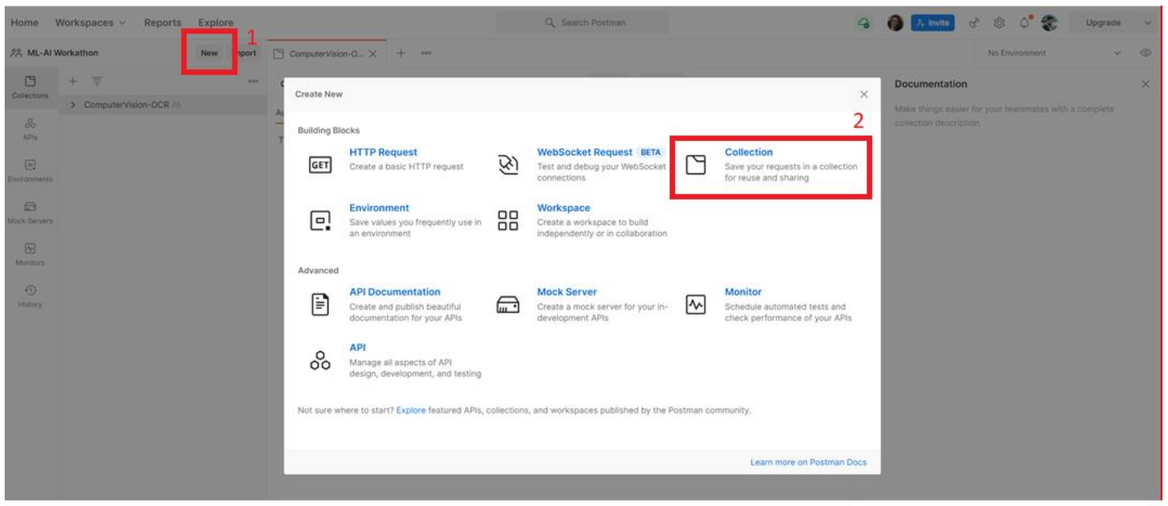
Significance :
The global variables will be created once and leveraged time & again, in each API request that we make through Postman.
This way, you will not have to hard code the Endpoint & Key for every request you make, thereby, making it more secure. This will also save time and effort.

Follow step 1 & 2 to add global variables for :

1. Endpoint – Paste the endpoint of the Multipurpose service account copied earlier
2. Key - Paste the Key of the Multipurpose service account copied earlier



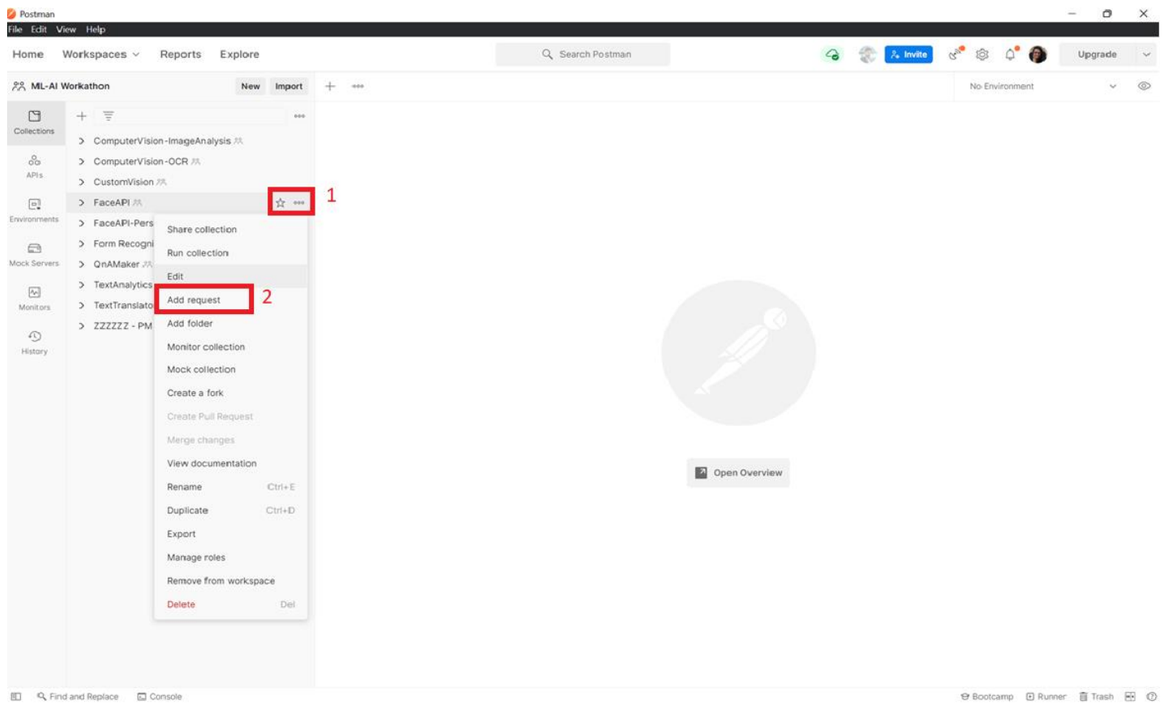

Once you have added the global variables, they will appear this way in your global variables section.



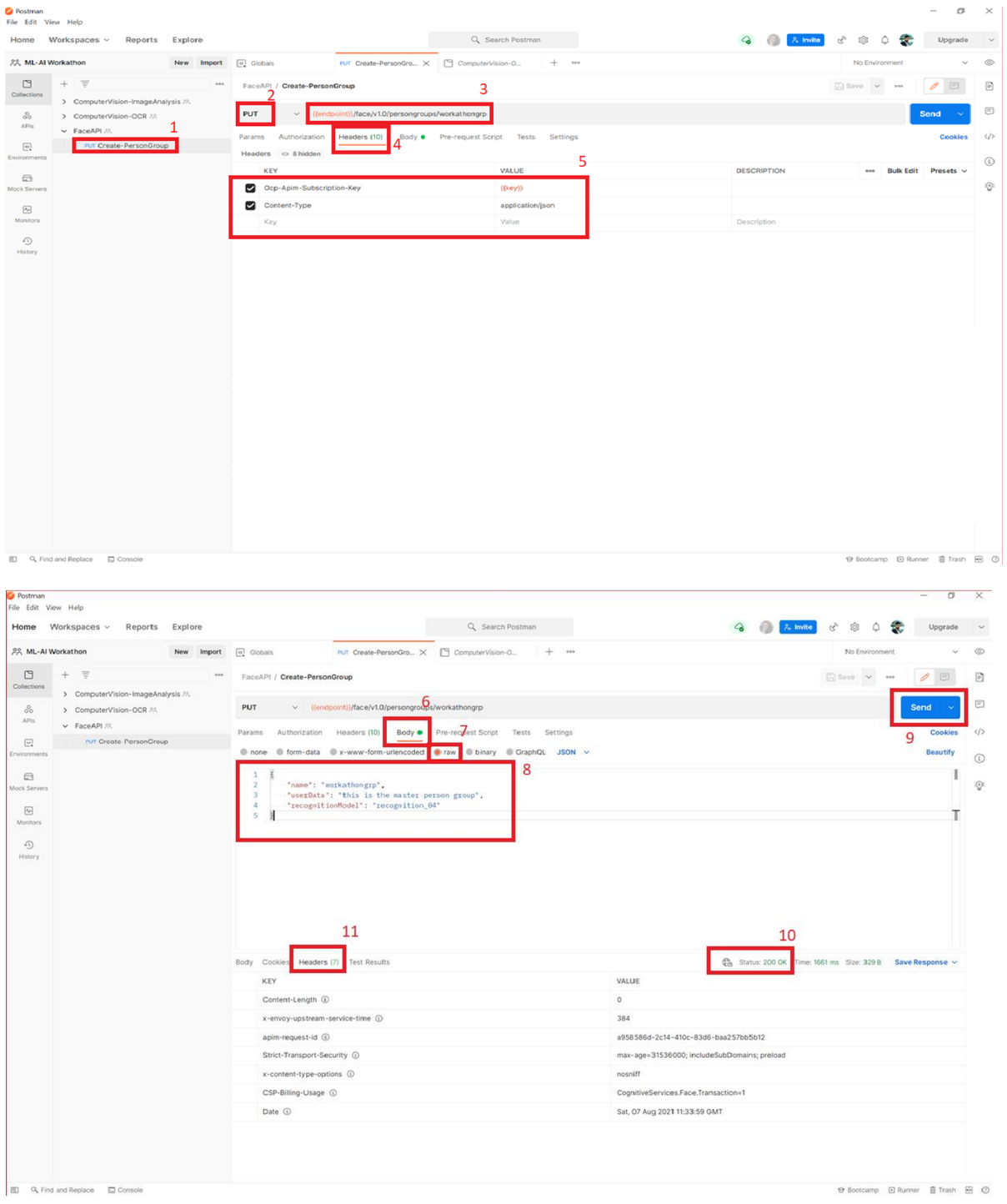

<u>Create new collection in Postman</u>

Open Postman > select New.
On the pop up select Collection.
Name the collection FaceAPI.

Collection is like a folder for managing the API call requests.




Once you have created the collection, follow steps 1 & 2, to create a new request.







## Create Person Group

This request upon successful execution will create a PersonGroup for us, with the name & Unique ID "workathongrp".

To create a Person Group, follow the sequential steps.

URL : {{endpoint}}/face/v1.0/persongroups/workathongrp

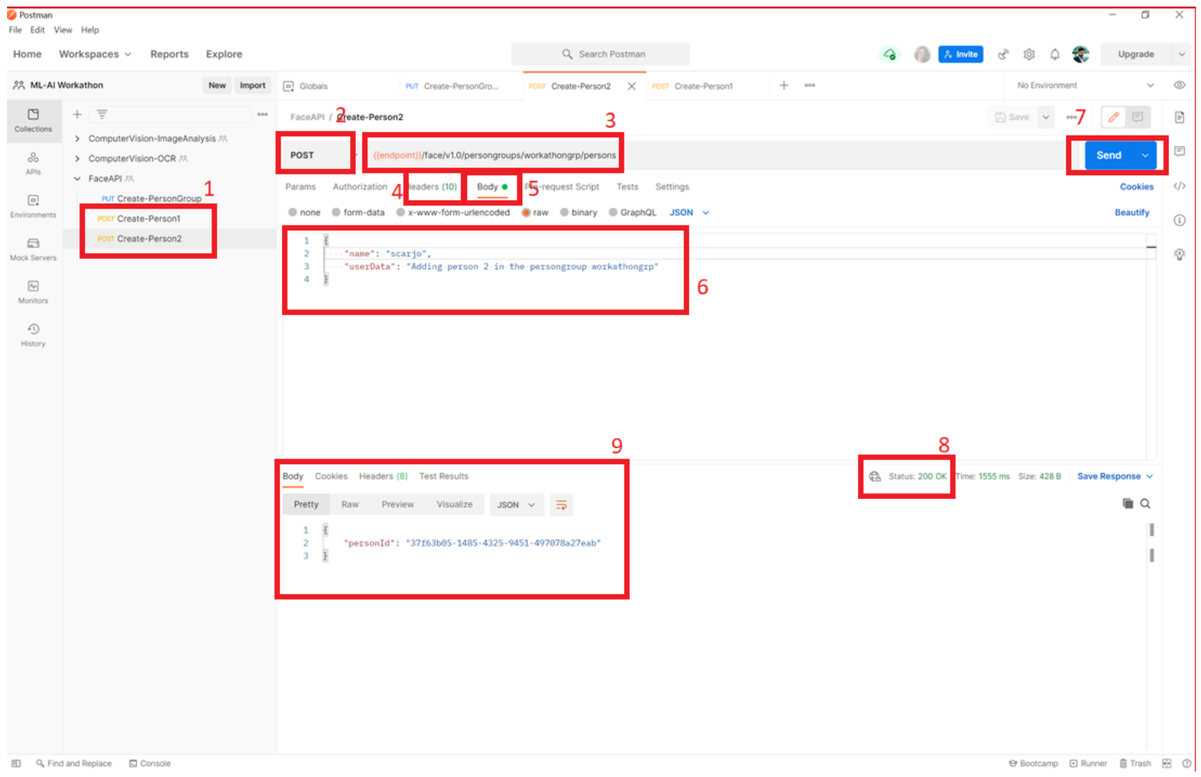
Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :

```
{
  "name": "workathongrp",
  "userData": "this is the master person group",
  "recognitionModel": "recognition_04"
}
```

## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables
2. Workathongrp : This is the Unique Person Group ID.
3. Note the recognitionModel key value pair in the body. This decides how the model will extract features for all the faces you upload to any Person object created in this Person Group. The default value is recognition_01, which is an older recognition model. Hence, we use recognition_04, since this provides highest accuracy.
4. After you execute the call, observe the status returned, as shown in step 10. This should reflect 200 OK.




## Create Person objects

We will create 2 persons with names keanu & scarjo respectively, under Person Group workathongrp created above.

To create a Person object, follow the sequential steps twice, once for each person:

URL : {{endpoint}}/face/v1.0/persongroups/workathongrp/persons

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :
For person 1 :

```
{
  "name": "keanu",
  "userData": "Adding person 1 in the persongroup workathongrp"
}
```

For person 2 :

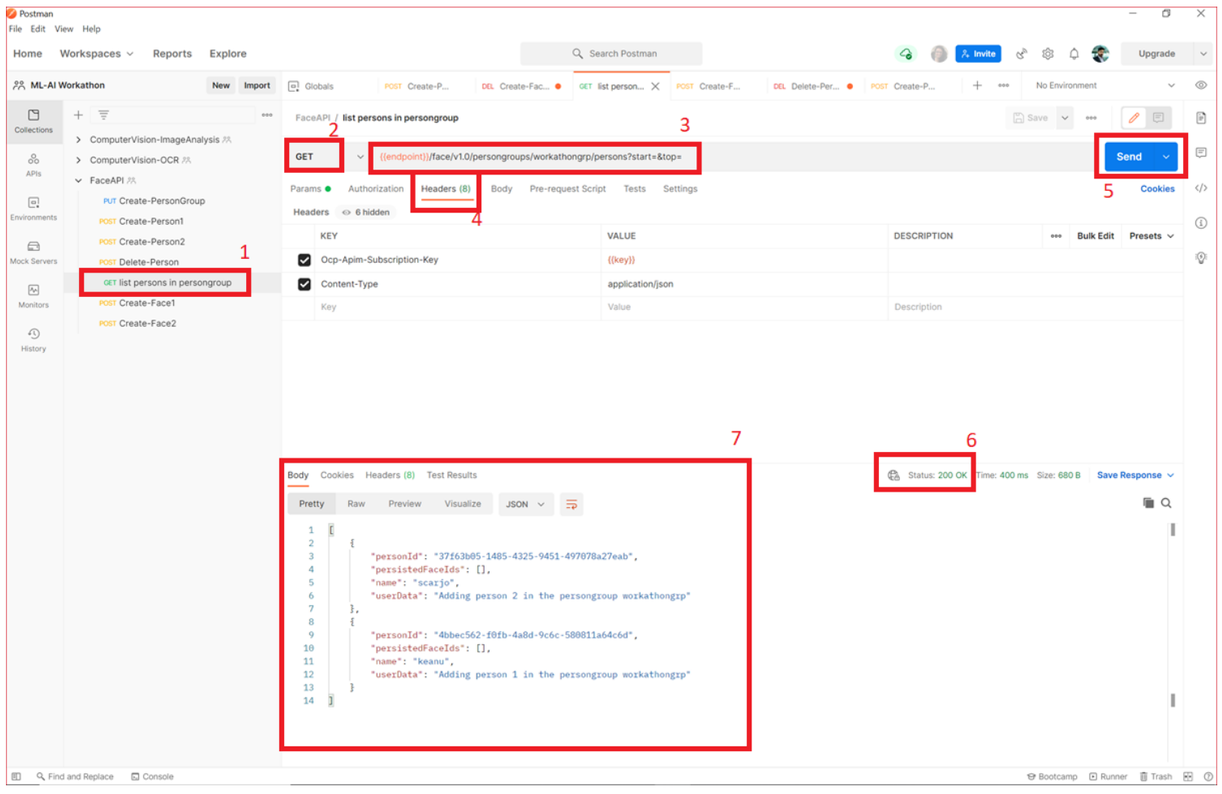
```
{
  "name": "scarjo",
  "userData": "Adding person 2 in the persongroup workathongrp"
}
```

## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables

2. After you execute the call, observe the status returned, as shown in step 8. This should reflect 200 OK.

3. On successful completion, the call will return personId.
   This Id will be unique for every person object added in the person group. We will leverage going forward, in order to add face images to a particular person.




## List the details of all the persons in the PersonGroup

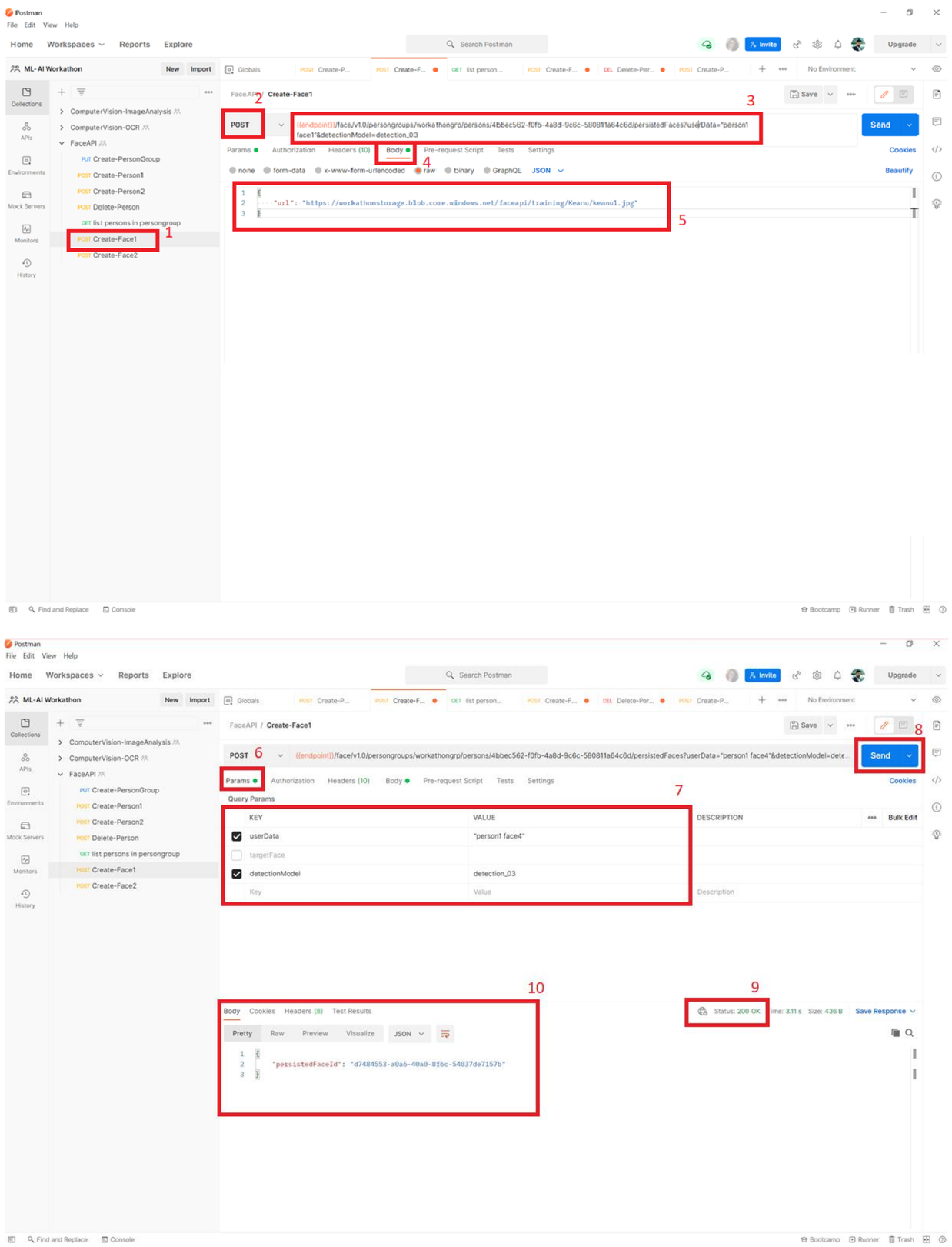
Any time you want to see the details of all Person objects added in a Person Group, you can leverage the List API.
The personId returned in this call will be leveraged in the next steps, to add faces to each Person object.

URL : {{endpoint}}/face/v1.0/persongroups/workathongrp/persons?start=&top=

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables
2. After you execute the call, observe the status returned, as shown in step 6. This should reflect 200 OK.
3. On successful completion, the call will return "personId", "persistedFaceIds", "name" and "userData" for each person in Person Group, 2 person objects in our case.
4. If you see a mismatch with highlighted step 7, please verify how you created the Person Group and Person Objects above.
5. "persistedFaceIds" is an empty array since we haven't added any faces to the Person Objects yet. We will do that in the next steps.




## Add faces to a person

We will add 4 faces each to Keanu & scarjo Person objects created above.

To add a face to a Person object, follow the sequential steps, once for face image to be added:

URL : {{endpoint}}/face/v1.0/persongroups/workathongrp/persons/4bbec562-f0fb-4a8d-9c6c-580811a64c6d/persistedFaces?userData="person1 face4"&detectionModel=detection_03
[replace highlighted text with personId you retrieved from the list person object step above]

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :
For person 1 face 1:
{
"url":"https://workathonstorage.blob.core.windows.net/faceapi/training/Keanu/keanu4.jpg"
}
[ This is the blob storage URL path for the image you want to add.
Replace the URL path if you used a different naming convention. If you followed our naming convention for creating containers & folders in blob, this URL will work]

Params:
userData : "person1 face1" [This is meta data for the face]
detectionModel : detection_03

## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables

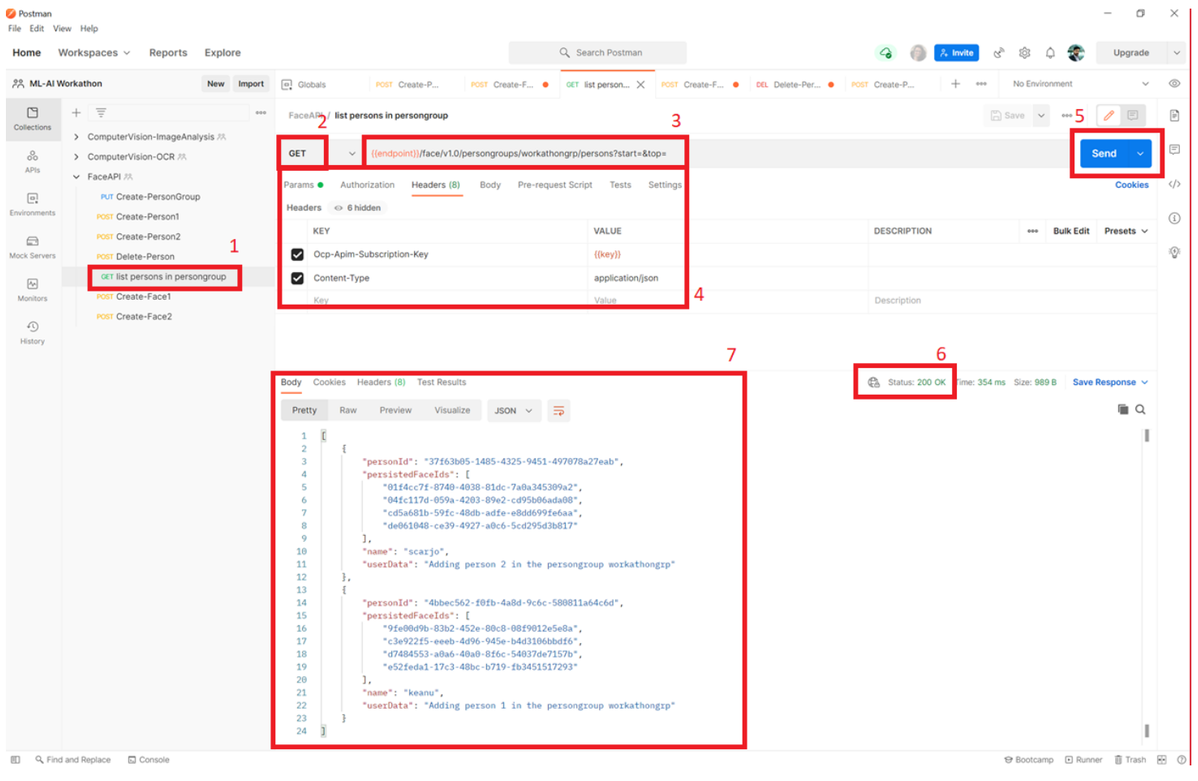
2. Observe the detectionModel parameter in Params section. This decides the features the model will extract for all the faces you upload to any Person object.
   The default value is detection_01, which is an older detection model. Hence, we use detection_03, since this provides highest accuracy, even on small or blurry faces.

3. After you execute the call, observe the status returned, as shown in step 9. This should reflect 200 OK.

4. On successful completion, the call will return persistedFaceId.
   This Id will be unique for every face added in the person object.

## Step Repetition

1. Since we are uploading 4 faces each to both Person Objects, we will repeat this API 4 times for each Person object. Hence, a total of 8 times.
2. Each time you add a new face to a Person, make sure to change the URL in Body and corresponding userData in Params section.
   [For Eg : "person1 face2" , "person1 face3", "person2 face1" and so on]
3. Once you have added all faces for Person 1, repeat the process for Person 2. Make sure to change the personId in the URL and make the add face call.




## List the details of all the persons in the PersonGroup

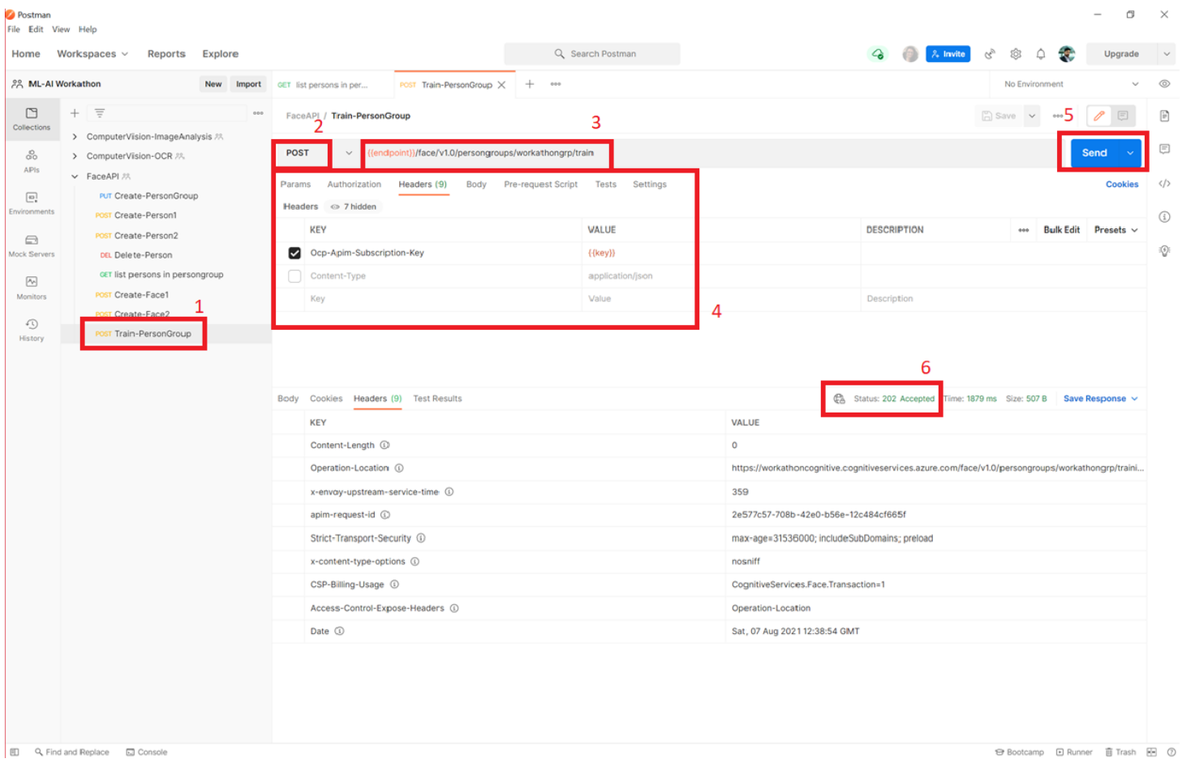
We will repeat the List Person details call we made earlier, once we have added all the faces to the corresponding Person objects.

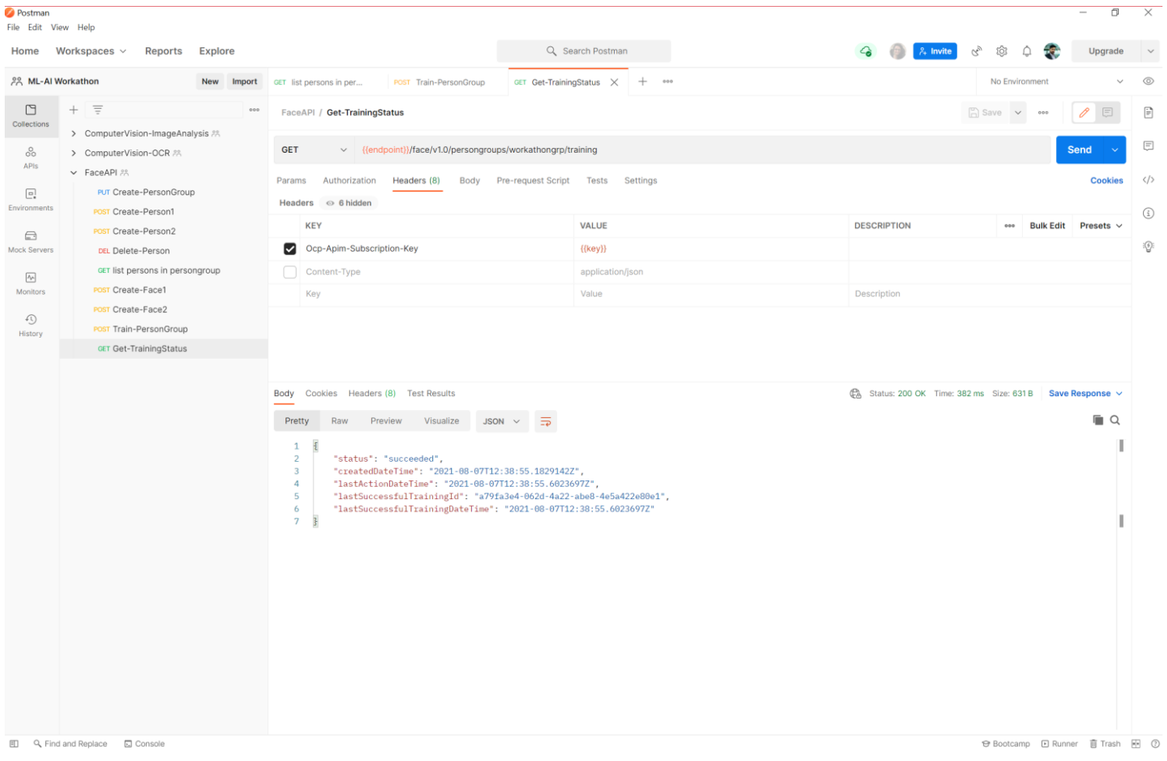
URL : {{endpoint}}/face/v1.0/persongroups/workathongrp/persons?start=&top=

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

## Significance of input & output

1. This time note the difference in the persistedFaceIds field returned in the result. You should see 4 IDs for each Person object.




## Train the person group

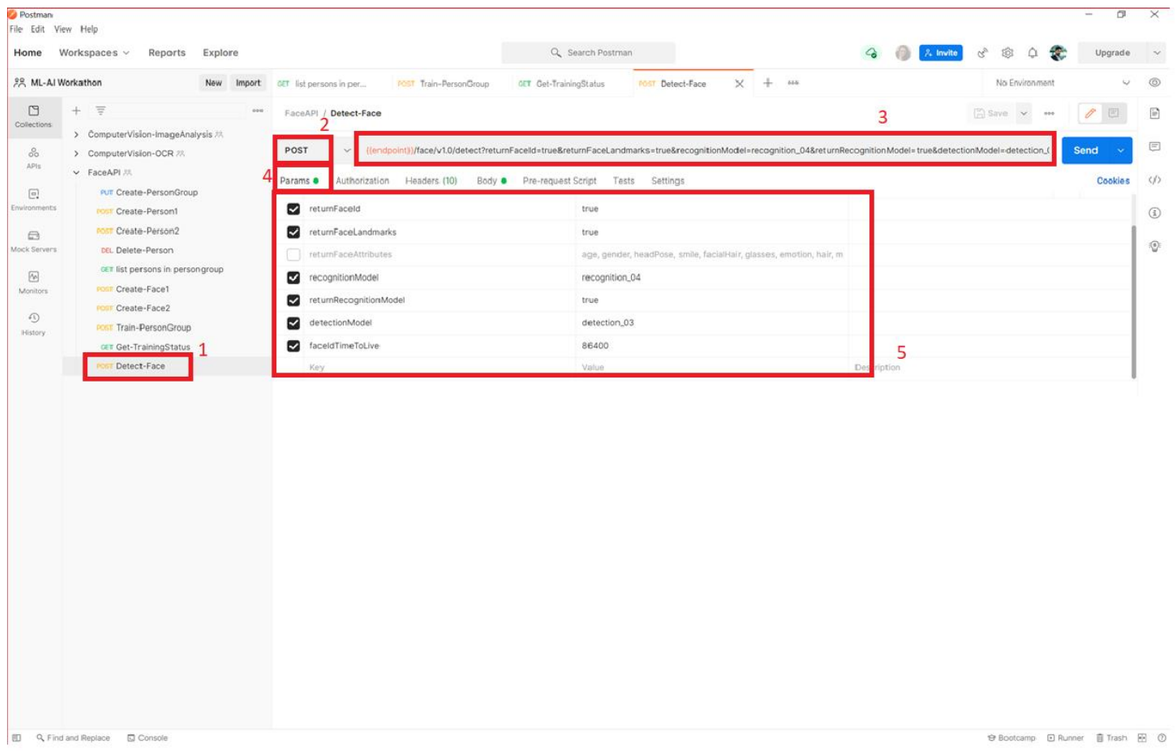
Once we have added all faces to the Person objects, we are ready to train the model.

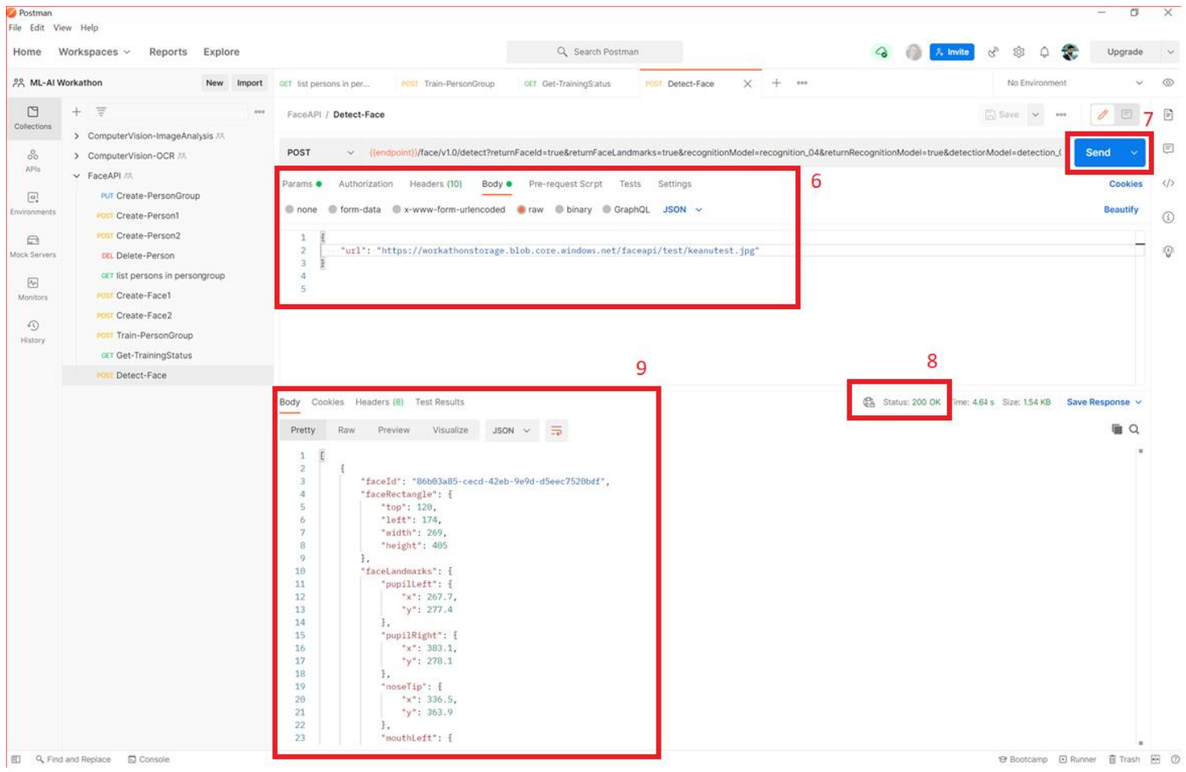
URL : {{endpoint}}/face/v1.0/persongroups/workathongrp/training

Headers :
Ocp-Apim-Subscription-Key : {{key}}

## Significance of input & output

1. We need to train a Person Group before we can use it for Identification use case. Only a trained Person Group can be used for identification.
2. The training task is an asynchronous task. Training time depends on the number of person entries, and their faces in a person group. It could be several seconds to minutes.




## Detect Face

This API has 2 use cases :

1. Analyse functionality of Face API – The left portion of the Function Architecture Diagram at the beginning of the workshop.
   The result for this API returns a Unique faceId for all the faces detected in the input image, coordinates of all the faces, their attributes and landmarks. These concepts were explained at the beginning of the workshop.

2. As a preliminary step in Verification & Identification API calls
   The faceId returned by this API call is required as an input for the Verification & Identification API calls. Hence, this is a mandatory step in those use cases. We will perform Verification & Identification in the following steps.

URL : {{endpoint}}/face/v1.0/detect

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :
```
{
   "url": "https://workathonstorage.blob.core.windows.net/faceapi/test/Johanssontest.jpg"
}
```
[ This is the blob storage URL path for the image you want to detect.
Replace the URL path if you used a different naming convention. If you followed our naming convention for creating containers & folders in blob, this URL will work]

Params:
returnFaceId : true
returnFaceLandmarks : true
recognitionModel : recognition_04
detectionModel : detection_03
faceIdTimeToLive : 86400
returnRecognitionModel : true

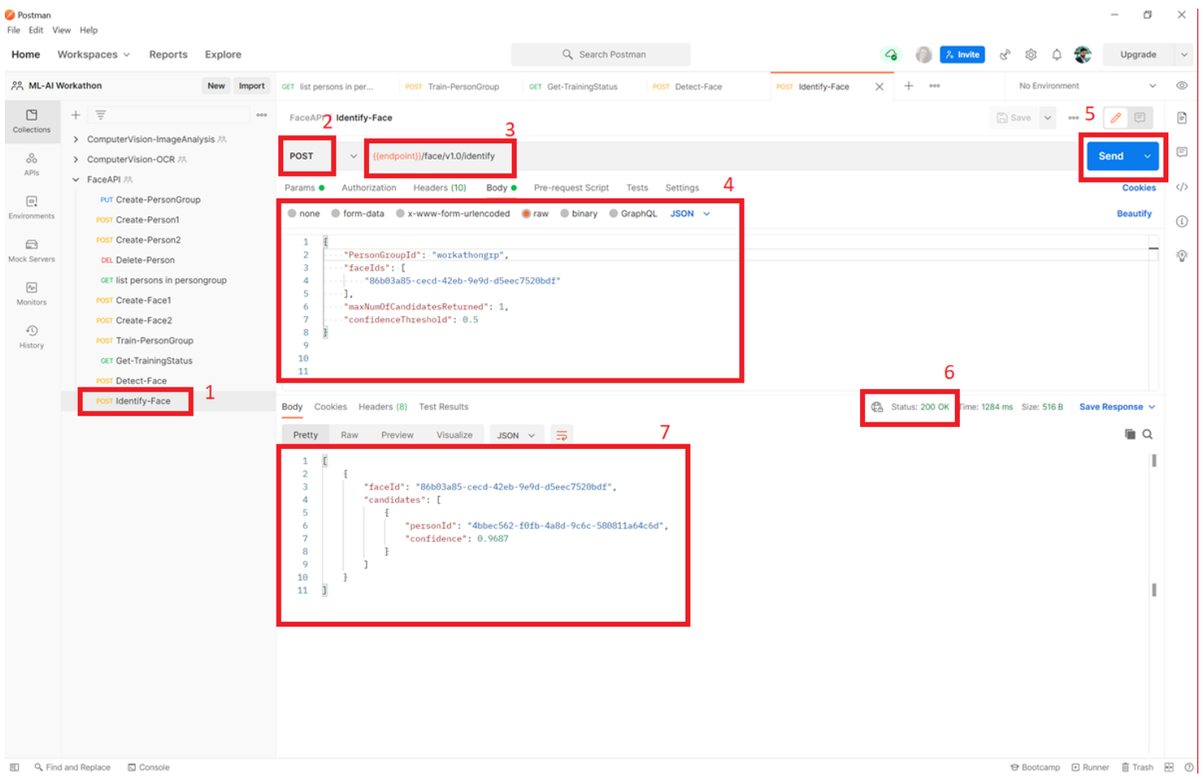
### Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables

2. Observe the detectionModel parameter in Params section. This should be consistent with the detectionModel Param used earlier. Similarly, for recognitionModel.
   If you don't maintain consistency, it will affect the accuracy of your model.

3. After you execute the call, observe the status returned, as shown in step 8. This should reflect 200 OK.

4. Observe the body section of the result returned.




## Identify Face

In day-to-day terms, this API solves the problem statement “Whom does this face belong to?” This call is used for a one-to-many face matching. This returns the Person with whom the input face matched, out of all the Persons in a Person Group

URL : {{endpoint}}/face/v1.0/identify

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :

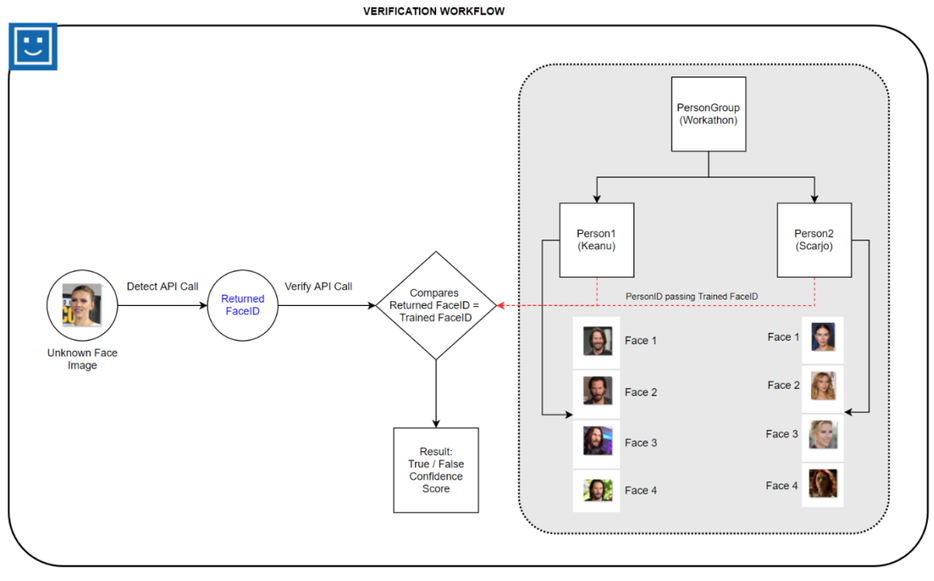
```
{
  "PersonGroupId": "workathongrp",
  "faceIds": [
    "86b03a85-cecd-42eb-9e9d-d5eec7520bdf"
  ],
  "maxNumOfCandidatesReturned": 1,
  "confidenceThreshold": 0.5
}
```

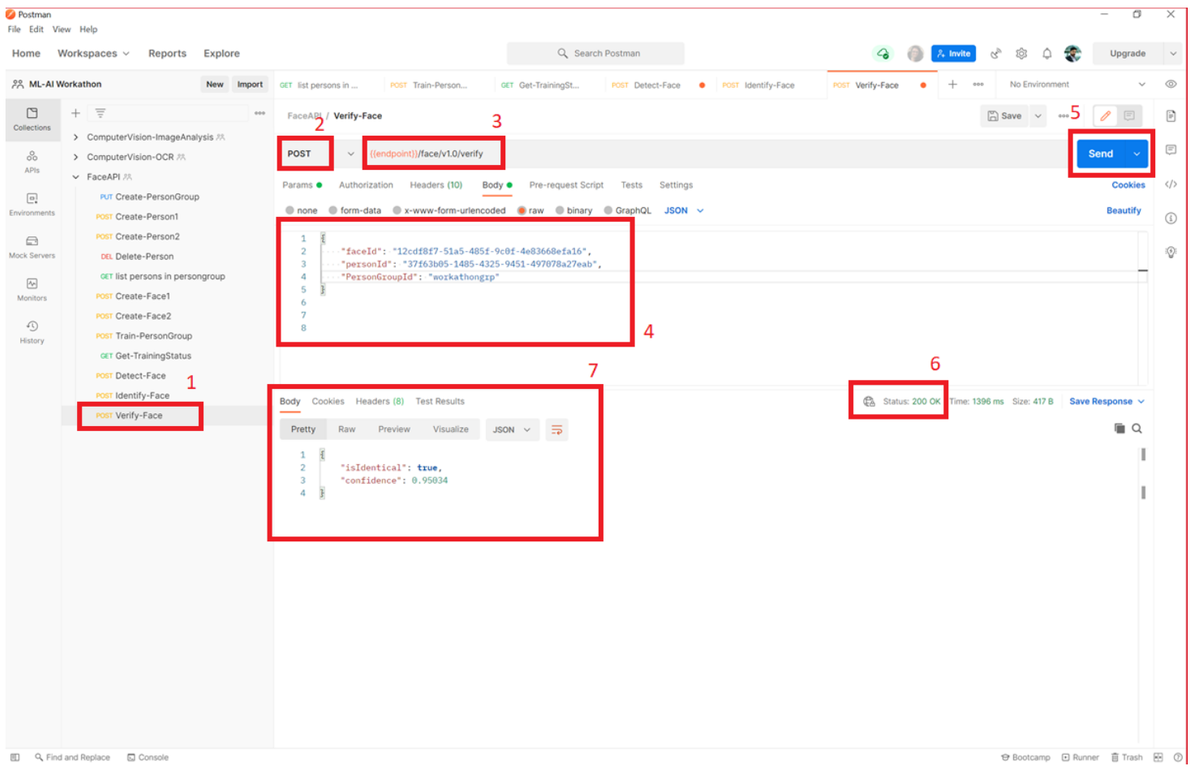
## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables
2. PersonGroupId value in the Body states the Person Group you want to identify the input face against.
3. faceIds is an array of input faces you want to identify. These faceIds are obtained using the Detect Face API call explored above.
4. The result returns PersonId of the Person to which the model found a match for the input face. maxNumOfCandidatesReturned as specified in input body are returned.
5. Confidence value in the result signifies how confident the model is in saying that the input face belongs to that person.
6. The call will return all faces that have a confidence greater than confidenceThreshold parameter in body input and will return a maximum number of faces as specified by maxNumOfCandidatesReturned.

## Verification use case in Azure Face Service

The objects and hierarchy (Person Group, Person, Faces) we created above for the Identification use case will be used by the Verification use case.
The only difference lies in the face that you don’t necessarily need to train a Person Group to leverage it for Verification purposes.






## Verify Face

In day-to-day terms, this API solves the problem statement “Does this face belong to person X?”
This call is used for a one-to-one face matching. This returns a Boolean value for whether 2 faces belong to the same person or whether a new face belongs to a Person X in a Person Group.

URL : {{endpoint}}/face/v1.0/verify

Headers :
Ocp-Apim-Subscription-Key : {{key}}
Content-Type : application/json

Body :

```
{
  "faceId": "12cdf8f7-51a5-485f-9c0f-4e83668efa16",
  "personId": "37f63b05-1485-4325-9451-497078a27eab",
  "PersonGroupId": "workathongrp"
}
```

## Significance of input & output

1. {{endpoint}} , {{key}} : Values being picked from global variables
2. faceId is the unique ID of input faces you want to verify. These faceId is obtained using the Detect Face API call explored above.
3. personId is the unique ID of the Person you want to verify the input face against. You can fetch this from the List Person details API explored earlier.
4. PersonGroupId refers to the PersonGroup this Person object belongs to.
5. The call returns True if the confidence of matching is greater than 50%. You can leverage the confidence value to set a higher or lower threshold in your use case, to consider if the faces matched.

## Homework

Like we created the Person Group & Persons and added Faces to each Person object for the Identification & Verification scenarios, you can try implementing the use case for Find Similar and Group functionality offered by Azure Face Service.

Hints

1. Create a FaceList
2. Add faces to the FaceList
3. Call the Find Similar API
4. Call the Group API (You don’t need any pre created objects for this API)

## Additional recommended resources

| Service Name | Category | Links |
|---|---|---|
| Face Service | Programming Language | C#, Javascript, Python, Node.js, Go |
| | Tiers | Free (Not for Production), Standard \| (Differences between Tiers) |
| | Pricing | https://azure.microsoft.com/en-in/pricing/details/cognitive-services/face-api/ |
| | Limits | |
| | Language Support | Not applicable |
| | Sample Apps | FamilyNotes, ReactApp |
| | Regional Availability & Support | https://azure.microsoft.com/en-us/global-infrastructure/services/?products=cognitive-services&regions=all |
| | SLAs for Cognitive Services | https://azure.microsoft.com/en-in/support/legal/sla/cognitive-services/v1_1/ |
| | Compliance & Certificates | https://azure.microsoft.com/en-us/support/legal/cognitive-services-compliance-and-privacy/ |
| | Cognitive Services Updates | https://azure.microsoft.com/en-us/updates/?product=cognitive-services |

Security best practices

1. Azure Cognitive Services security
2. Networking
3. Authentication
4. Key Management
5. Data loss prevention
6. Azure security baseline
7. Regulatory Compliance controls

Responsible AI being a part of best practices, we encourage you to read this.

Face Service Documentation

API & Error references