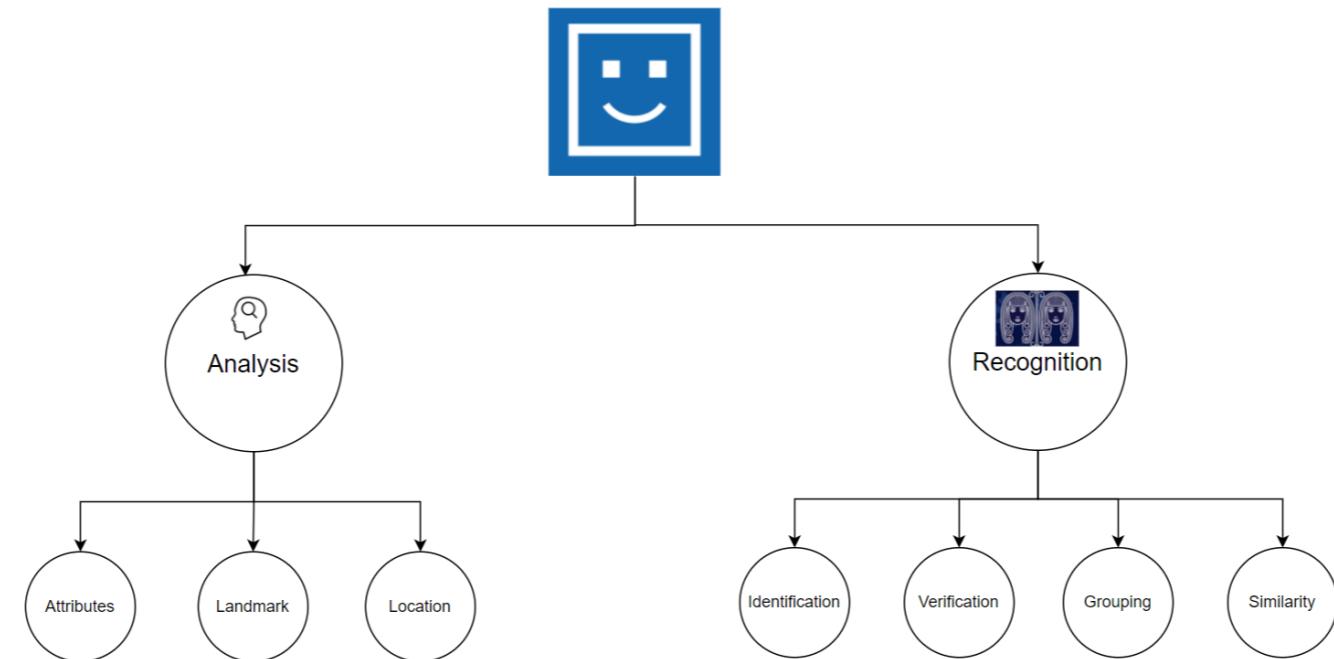


<b>Document Name</b>	HOL – Azure Face Service v1.0
<b>Author</b>	Shiva S Tomar & Anupreet Kaur
<b>Reviewer</b>	
<b>Executive Summary</b>	Azure Cognitive APIs enable the developers of all skill levels to add human intelligence in their applications. The services are designed for developers interested 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.
<b>Purpose</b>	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. Once you complete these labs, you'll go from <b>Zero to Hero</b> on the respective Azure Cognitive service and should be able to <b>Demo, Develop and Deploy</b> 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.
<b>Intent of Guide</b>	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. 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 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

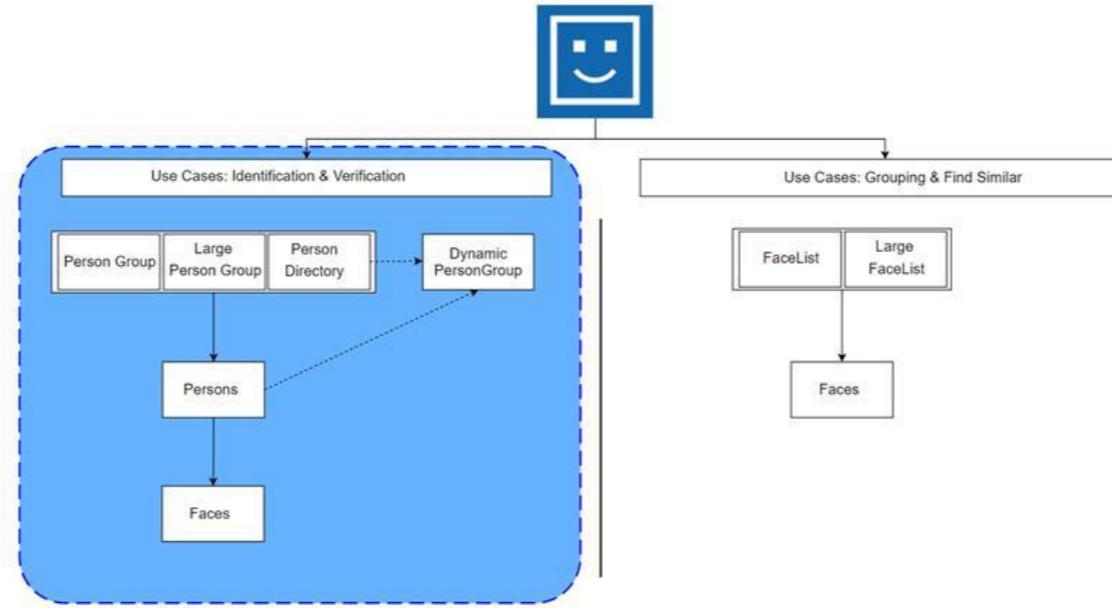
Like we saw in the functional architecture diagram above, Face Recognition feature has 4 functionalities in it -

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.

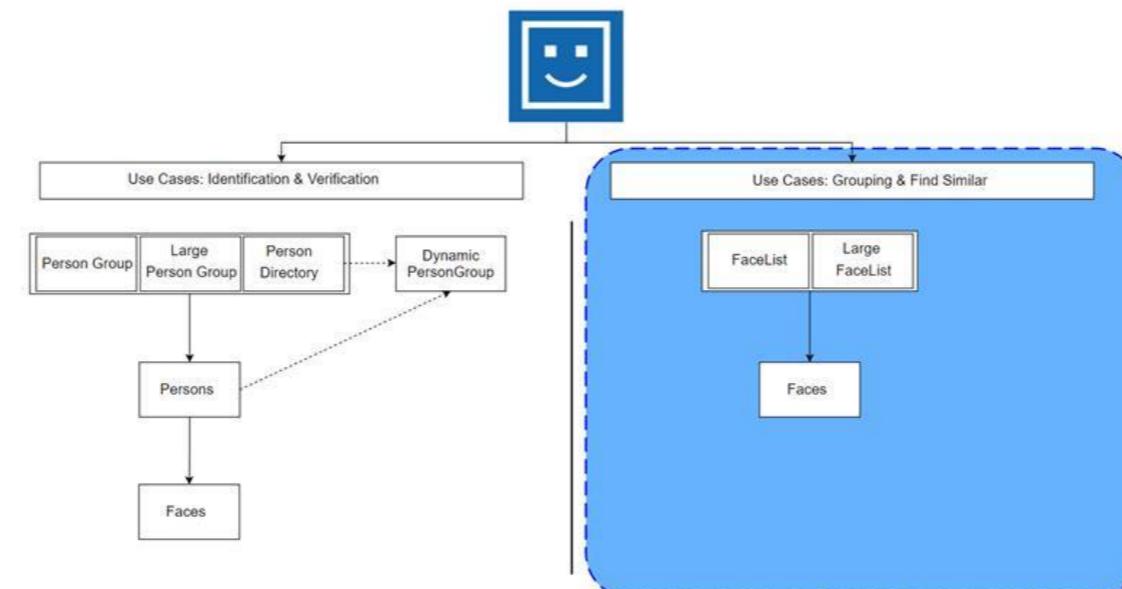
#### Concepts & Terminologies

\*\* 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 persistedFacId, 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.

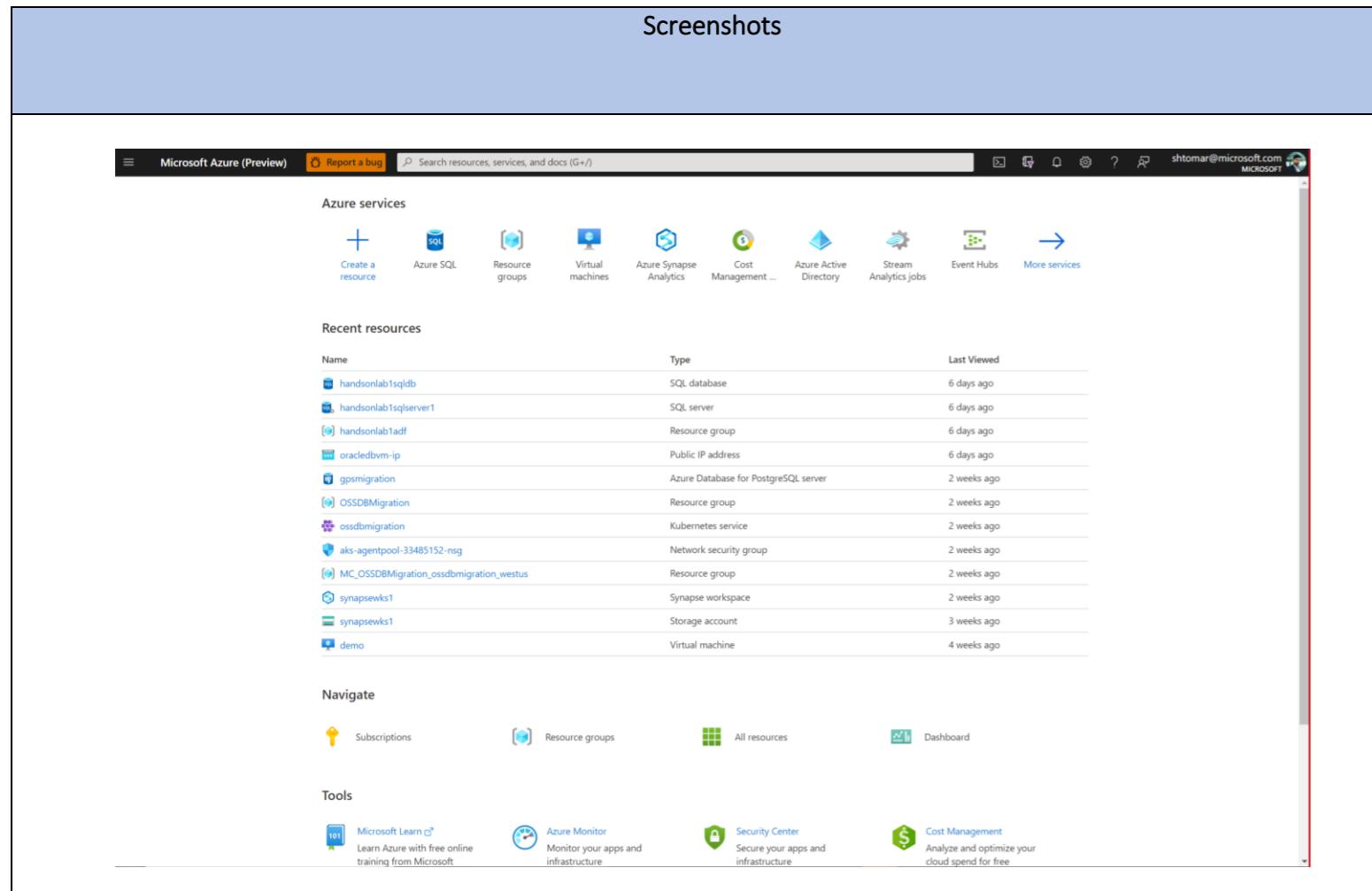
1. **Face** : This represents a single face that is stored as part of your model creation. Each face is associated with a unique persistedFacId, which is returned by the API when you add a face object in your model collection. This is the same as the Face object in Verification & Identification object 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
 The screenshot shows the Microsoft Azure (Preview) portal. At the top, there is a navigation bar with links for 'Report a bug' and 'Search resources, services, and docs (G+/-)'. On the right, it shows the user's email 'shtomar@microsoft.com' and a Microsoft logo. Below the navigation bar, the 'Azure services' section is visible, featuring icons for Create a resource, Azure SQL, Resource groups, Virtual machines, Azure Synapse Analytics, Cost Management, Azure Active Directory, Stream Analytics jobs, Event Hubs, and More services. Under 'Recent resources', there is a table listing various Azure resources with their names, types, and last viewed times. The table includes entries for handsonlab1sqldb (SQL database), handsonlab1sqlserver1 (SQL server), handsonlab1adf (Resource group), oracleidvm-ip (Public IP address), gpmigration (Azure Database for PostgreSQL server), OSSDBMigration (Resource group), ossdbmigration (Kubernetes service), aks-agentpool-33485152-ms (Network service group), MC_OSSDBMigration_ossdbmigration_westus (Resource group), synapsewks1 (Synapse workspace), synapsewks1 (Storage account), and demo (Virtual machine). In the 'Tools' section at the bottom, there are links for Microsoft Learn, Azure Monitor, Security Center, and Cost Management.	Sign into your Azure Portal.

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text 'resource'. Below the search bar, a dropdown menu is open, showing various service categories like 'Azure services', 'Recent resources', and 'Resources'. The 'Resource groups' option is highlighted with a red box and the number '2' next to it. Other options in the dropdown include 'Resource Guards (Preview)', 'Resource Explorer', 'Resource Graph Explorer', 'Resource Graph queries', 'Resource management private links', 'Subscriptions', 'My resources', 'All resources', 'Bing Resources', and 'Resource Groups'.

## Create a Resource Group

Follow steps 1 & 2 to create a resource group.

The screenshot shows the 'Resource groups' list page in the Microsoft Azure portal. At the top left, there is a 'Create' button highlighted with a red box. The page displays a list of existing resource groups, each with a checkbox, a name, a subscription, and a location. The columns are 'Name', 'Subscription', and 'Location'. The 'Subscription' column lists various internal subscriptions like 'anu\_internalSubscription' and external ones like 'Microsoft Azure Internal Consumption'. The 'Location' column lists regions like 'Southeast Asia', 'Central India', 'East US', etc.

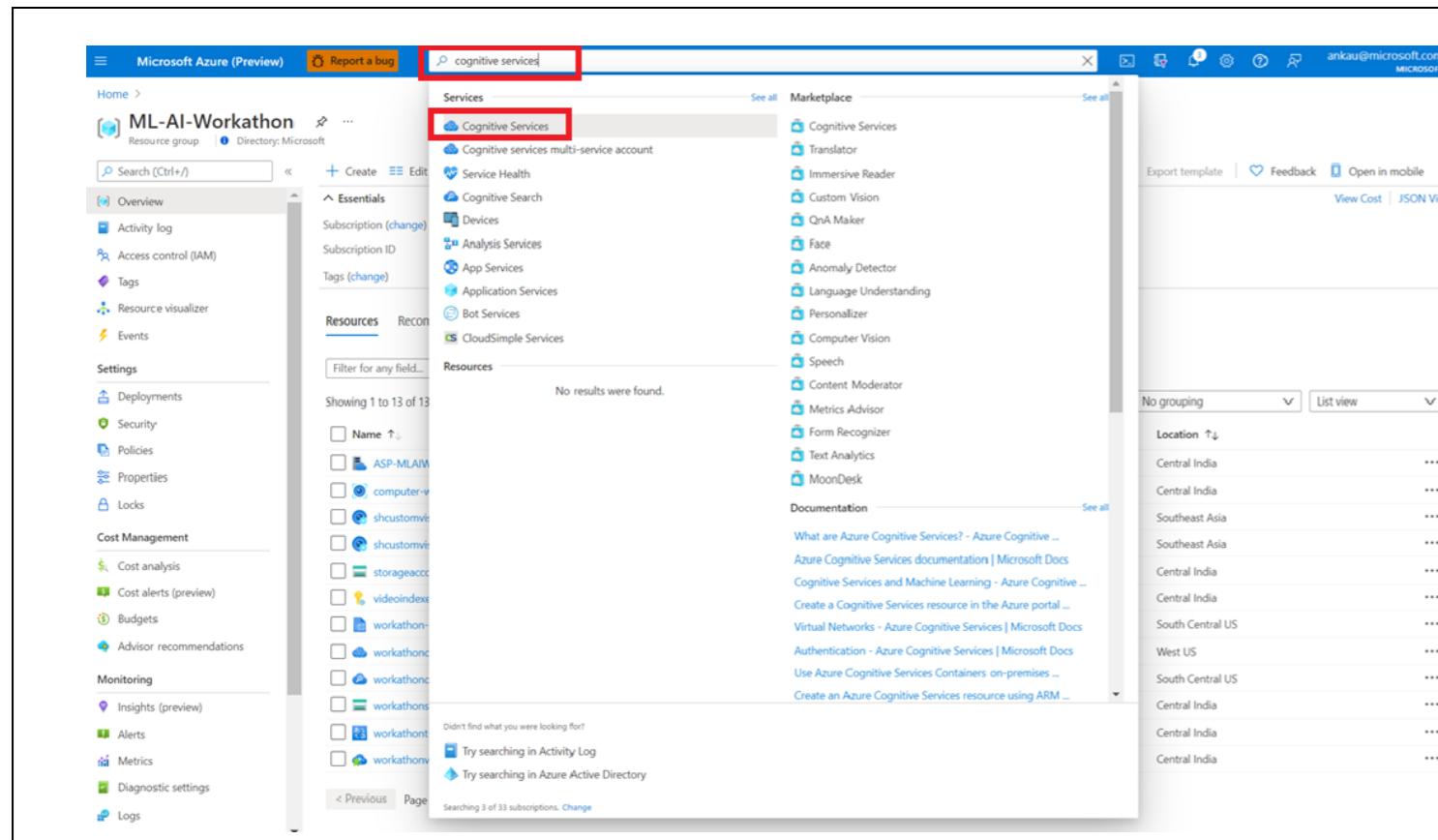
Click create to create a new resource group.

The screenshot shows the 'Create a resource group' wizard in the Microsoft Azure portal, specifically the 'Basics' step. It has three tabs: 'Basics', 'Tags', and 'Review + create'. The 'Basics' tab is selected. It contains fields for 'Project details' (Subscription: 'anu\_internalSubscription', Resource group: 'ML-AI-Workathon') and 'Resource details' (Region: '(Asia Pacific) Central India'). At the bottom, there are buttons for 'Review + create' (highlighted with a red box), '< Previous', and 'Next: Tags >'.

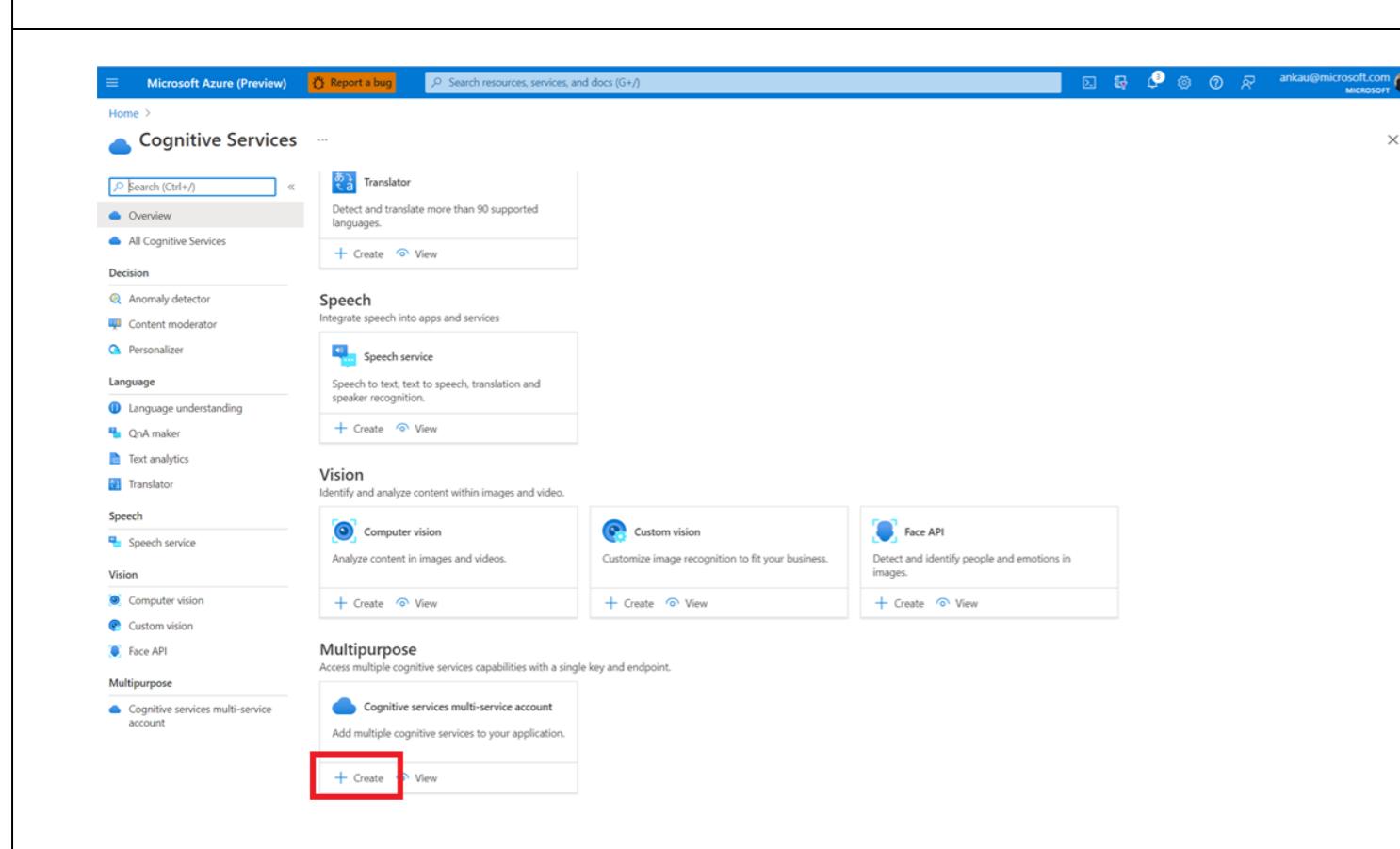
Enter the details –

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.

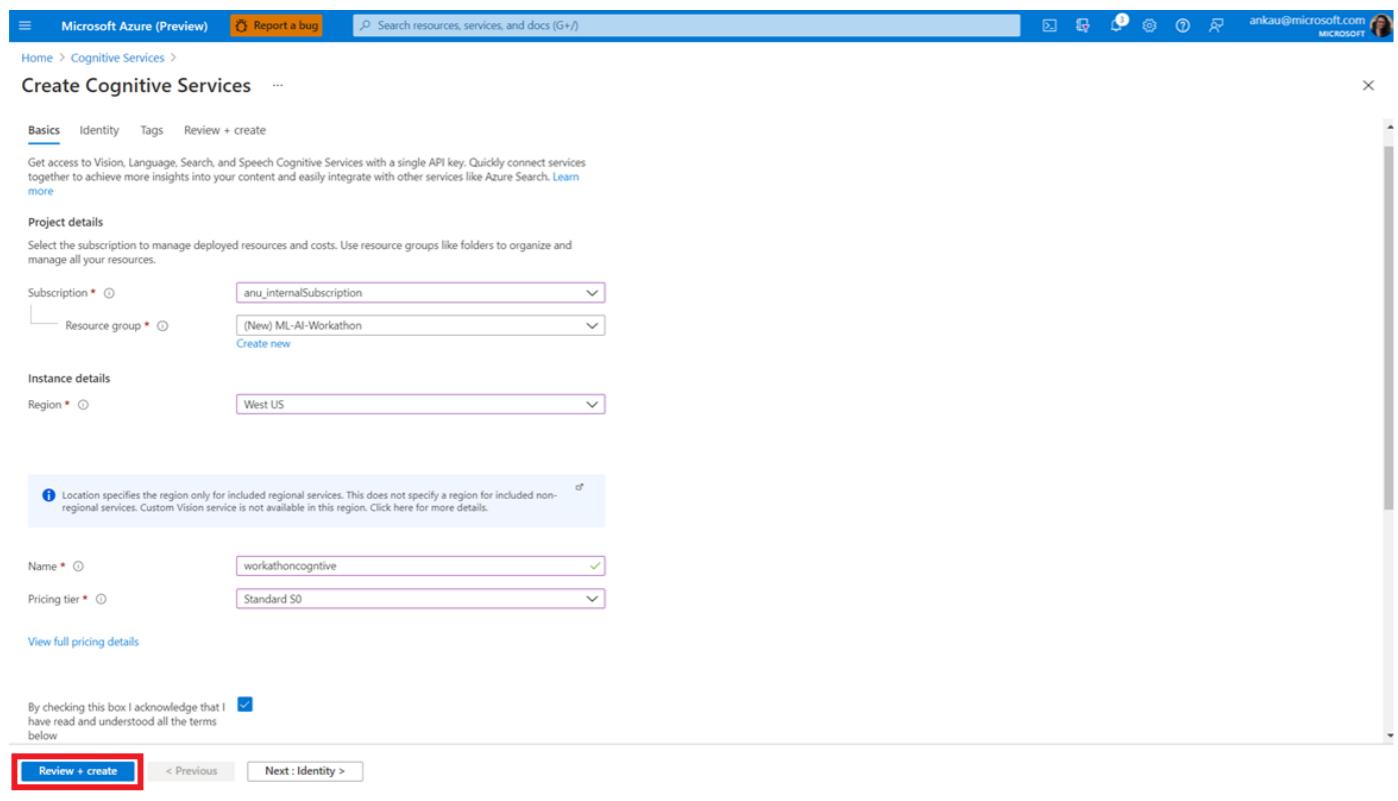


**Create a multipurpose cognitive service**

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

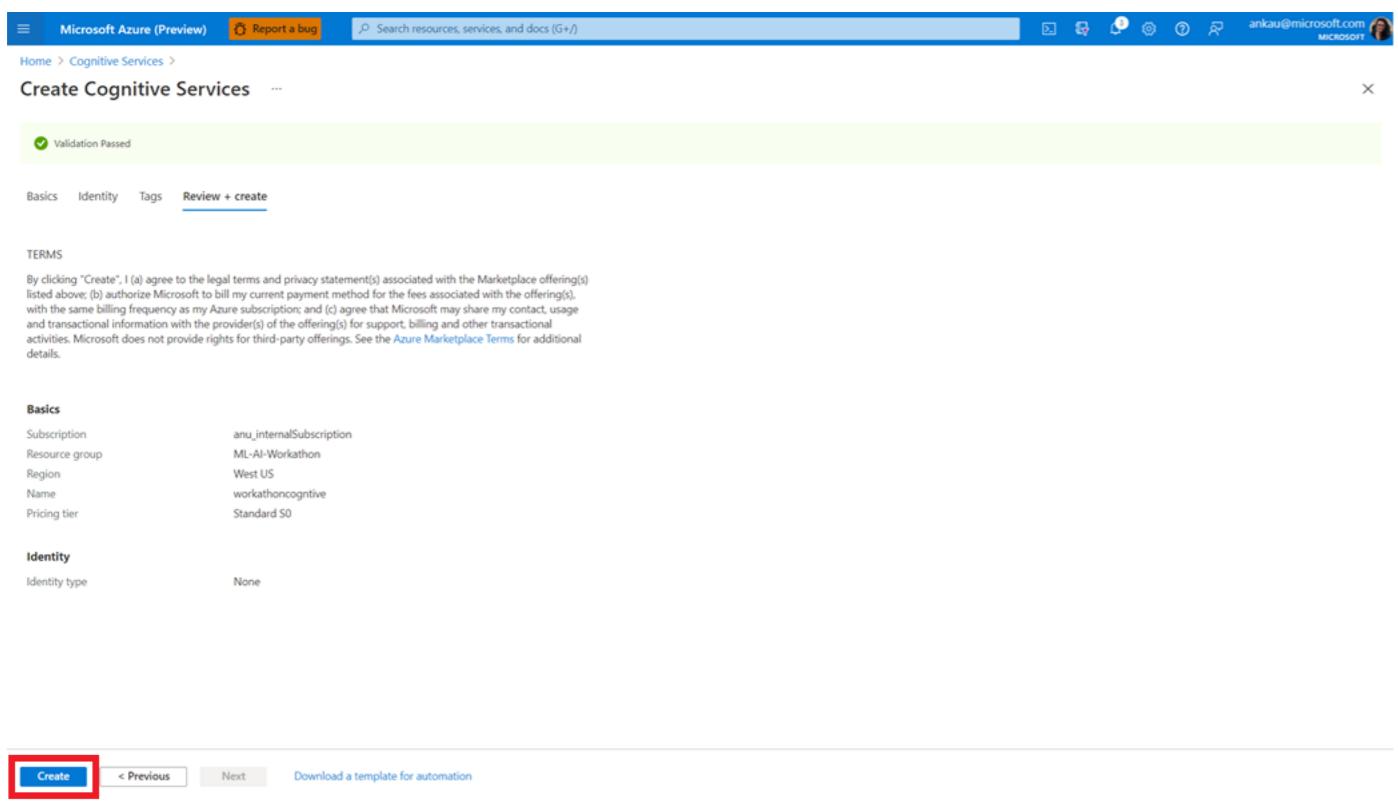
In this lab, we used a multipurpose Cognitive Service account since we would be learning about all the above-mentioned services. However, you can also spin up individual services to execute these labs or for your development / production scenarios. The only difference is spinning up individual services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

The screenshot shows the Microsoft Azure Deployment Overview page for a resource named "Microsoft.CognitiveServicesAllInOne-20210821163837". The main message is "Your deployment is complete". It provides deployment details: Deployment name: Microsoft.CognitiveServicesAllInOne-20210821... Start time: 8/21/2021, 4:43:51 PM; Subscription: arkaus@gmailSubscription; Resource group: ML-AI-Workathon. A "Go to resource" button is highlighted with a red box. To the right, there are links for Security Center, Free Microsoft tutorials, Work with an expert, and Cost Management.

The screenshot shows the Microsoft Azure Cognitive Services Quick start page for a resource named "workathoncognitive". The left sidebar shows "Resource Management" with "Keys and Endpoint" highlighted with a red box. The main content area has three numbered steps: 1. Grab your keys and endpoint (with a note about subscription keys); 2. Get an overview of what you can do with the Cognitive Services (with links to Documentation, Courses, and Community); 3. Get Started with the Cognitive Services (with a list of services like Computer Vision, Content Moderator, etc.).

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

### Copy keys & endpoints

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.

Microsoft Azure (Preview) | Report a bug | Search resources, services, and docs (G+) | ankaus@microsoft.com

Home > Microsoft.CognitiveServicesAllInOne-20210821163837 > workathoncognitive

workathoncognitive | Keys and Endpoint | Directory: Microsoft

Search (Ctrl+ /) | Regenerate Key1 | Regenerate Key2

Overview | Activity log | Access control (IAM) | Tags | Diagnose and solve problems

Resource Management

- Quick start
- Keys and Endpoint**
- Pricing tier
- Networking
- Identity
- Billing By Subscription
- Properties
- Locks

Monitoring

- Alerts
- Metrics
- Diagnostic settings
- Logs

Automation

- Tasks (preview)
- Export template

Support + troubleshooting

These keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

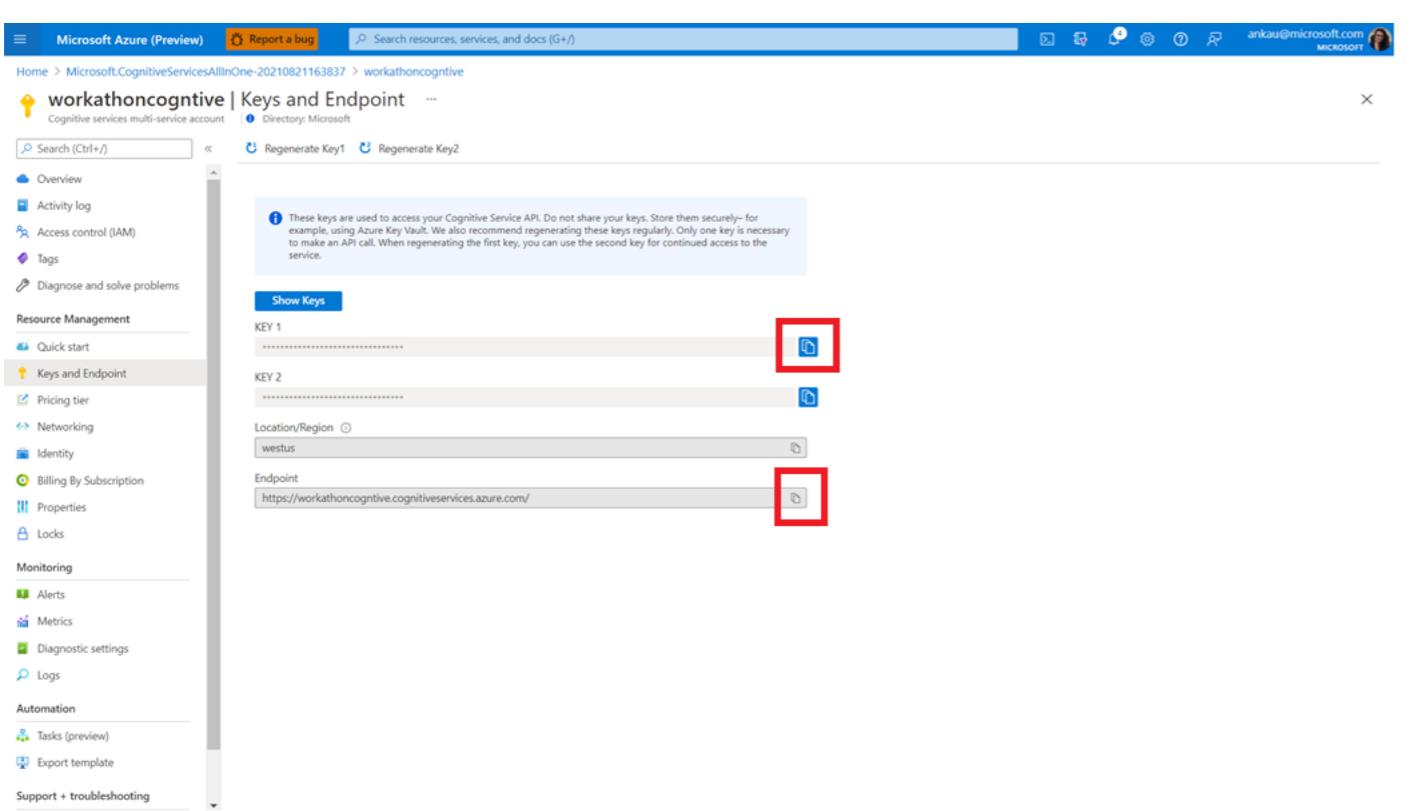
Show Keys

KEY 1

KEY 2

Location/Region: westus

Endpoint: https://workathoncognitive.cognitiveservices.azure.com/



Microsoft Azure (Preview) | Report a bug | Storage | 1 | shtomar@microsoft.com

Home > ML-AI-Workathon >

Create a resource | See all | Marketplace | See all

Get Started | Recently created | Categories

AI + Machine Learning | Analytics | Blockchain | Compute | Containers | Databases | Developer Tools | DevOps | Identity | Integration | Internet of Things | IT & Management Tools | Media | Migration | Mixed Reality | Monitoring & Diagnostics | Networking | Security | Storage | Web

Storage accounts | 2 | Storage Explorer | Storage accounts (classic) | Storage Sync Services | Data Lake Storage Gen1 | Disk Pools | HPC caches | Disks (classic) | OS images (classic) | VM images (classic)

Azure Cosmos DB | Create | Docs | Function App | Create | Docs | SQL Database | Create | Docs | Storage account | Create | Docs | DevOps Start | Create | Docs | MS Learn | Web App | Create | Docs | MS Learn

Pure Cloud Block Store™ (Product Deployment) | Pure Cloud Block Store™ (subscription) | M365 Workplace Cloud Storage | Easy Intune Storage | Enterprise File Fabric

Documentation | See all

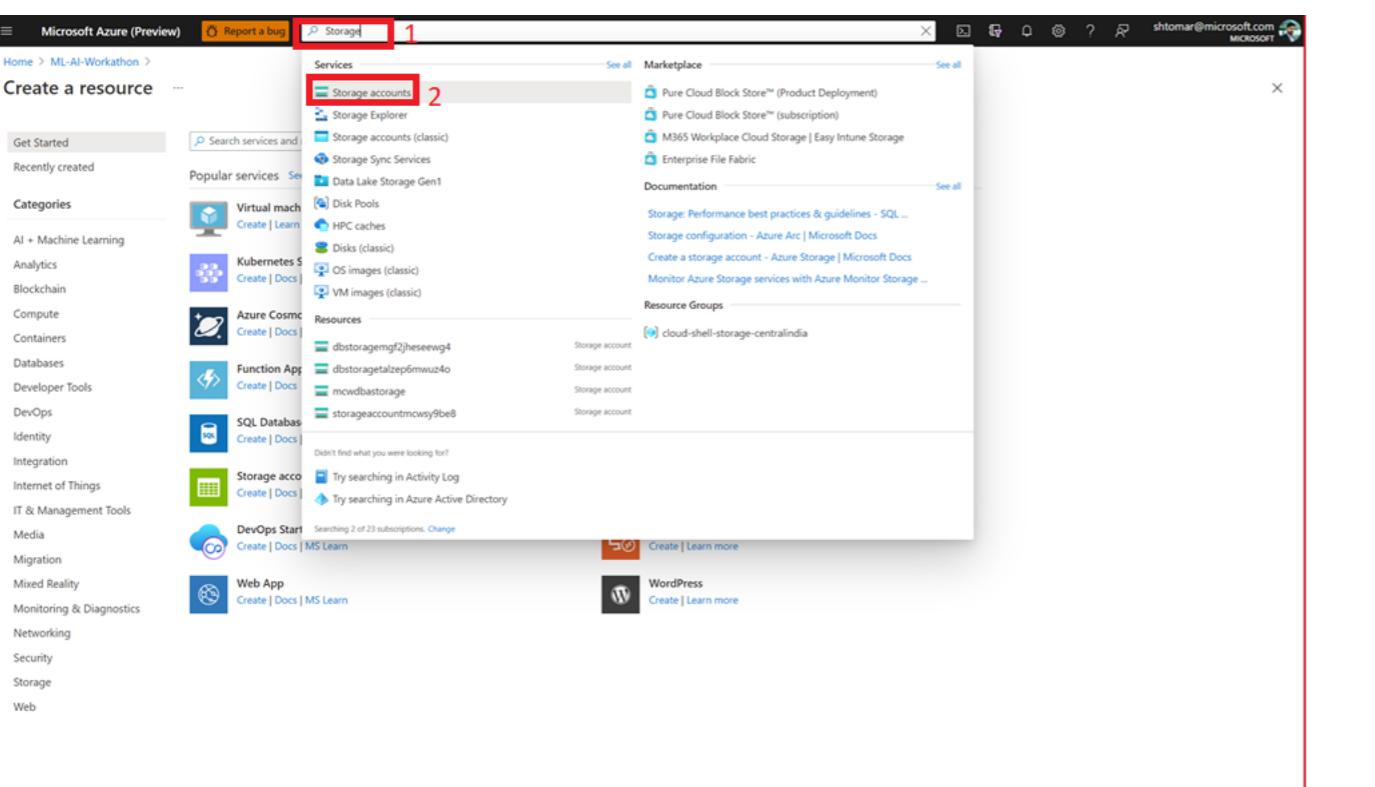
Storage: Performance best practices & guidelines - SQL ... | Storage configuration - Azure Arc | Microsoft Docs | Create a storage account - Azure Storage | Microsoft Docs | Monitor Azure Storage services with Azure Monitor Storage ...

Resource Groups

cloud-shell-storage-centralindia

dbstoragemgf2/heseevg4 | dbstoragealzepp6mwuz4o | mcwdbastorage | storageaccountmcwsy9be8

Didn't find what you were looking for? Try searching in Activity Log | Try searching in Azure Active Directory

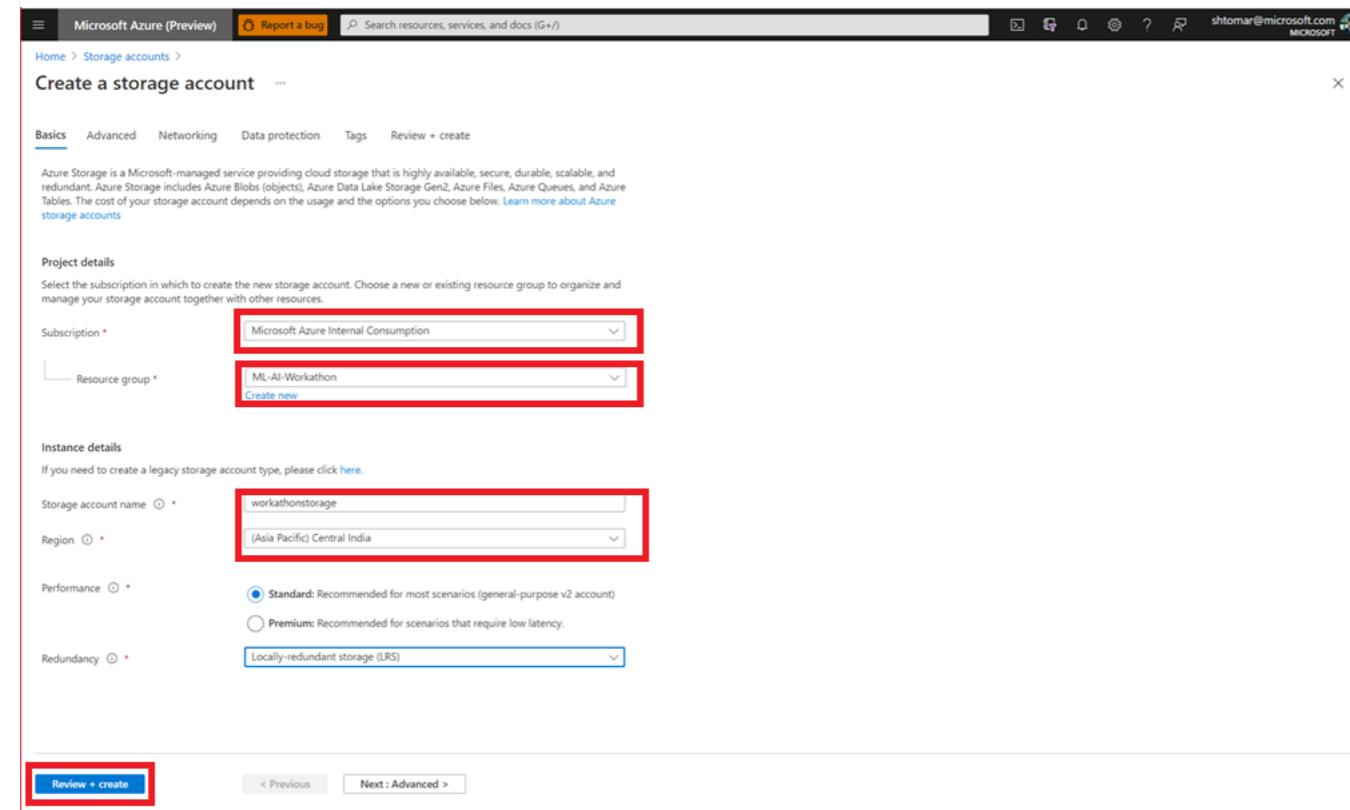


Copy the Key and Endpoint. Paste these in a notepad. You will leverage these at a later step, while setting up the global variables in Postman.

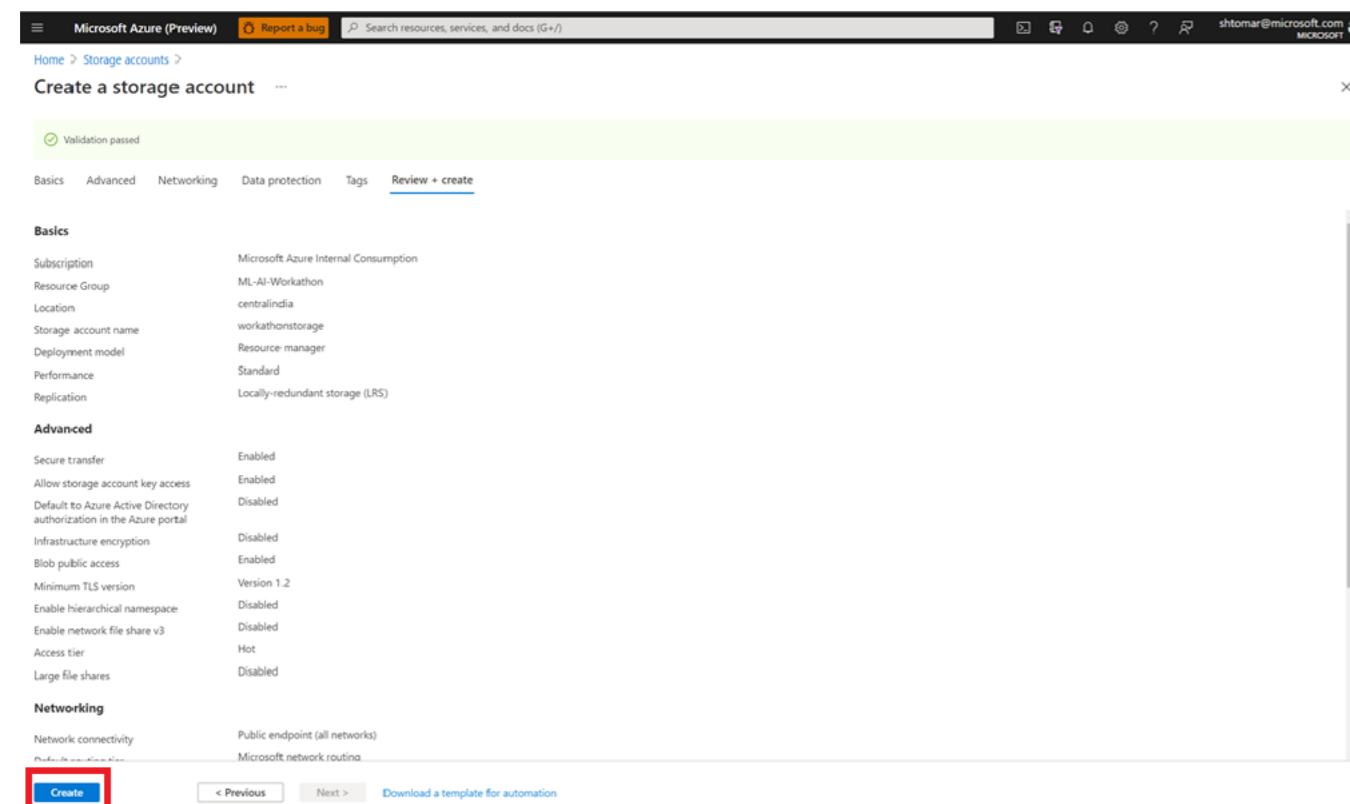
**Create Storage Account in Azure Portal**

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.

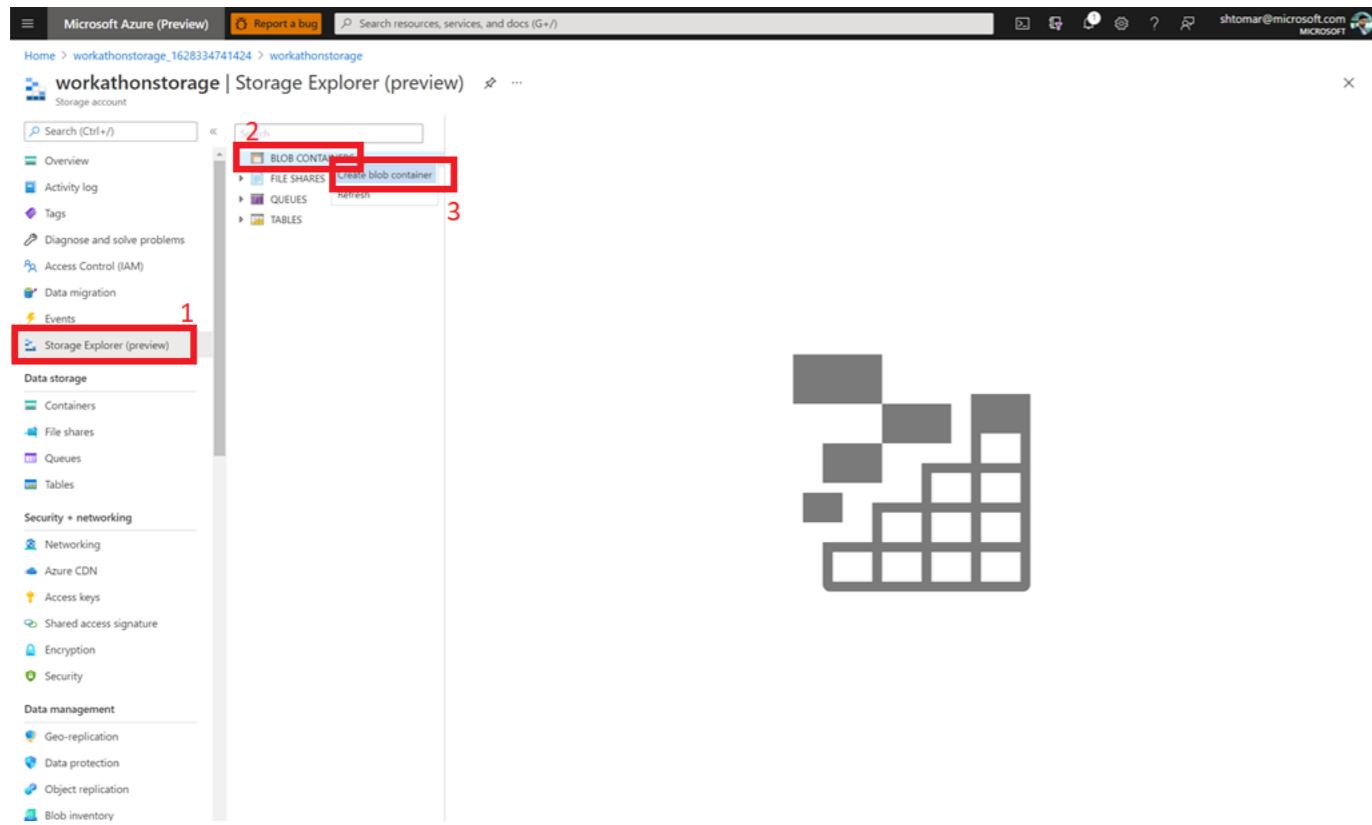
You can skip this step if you already have a Storage Account in place.



Enter the details highlighted in the screenshot.  
Click Review + Create.

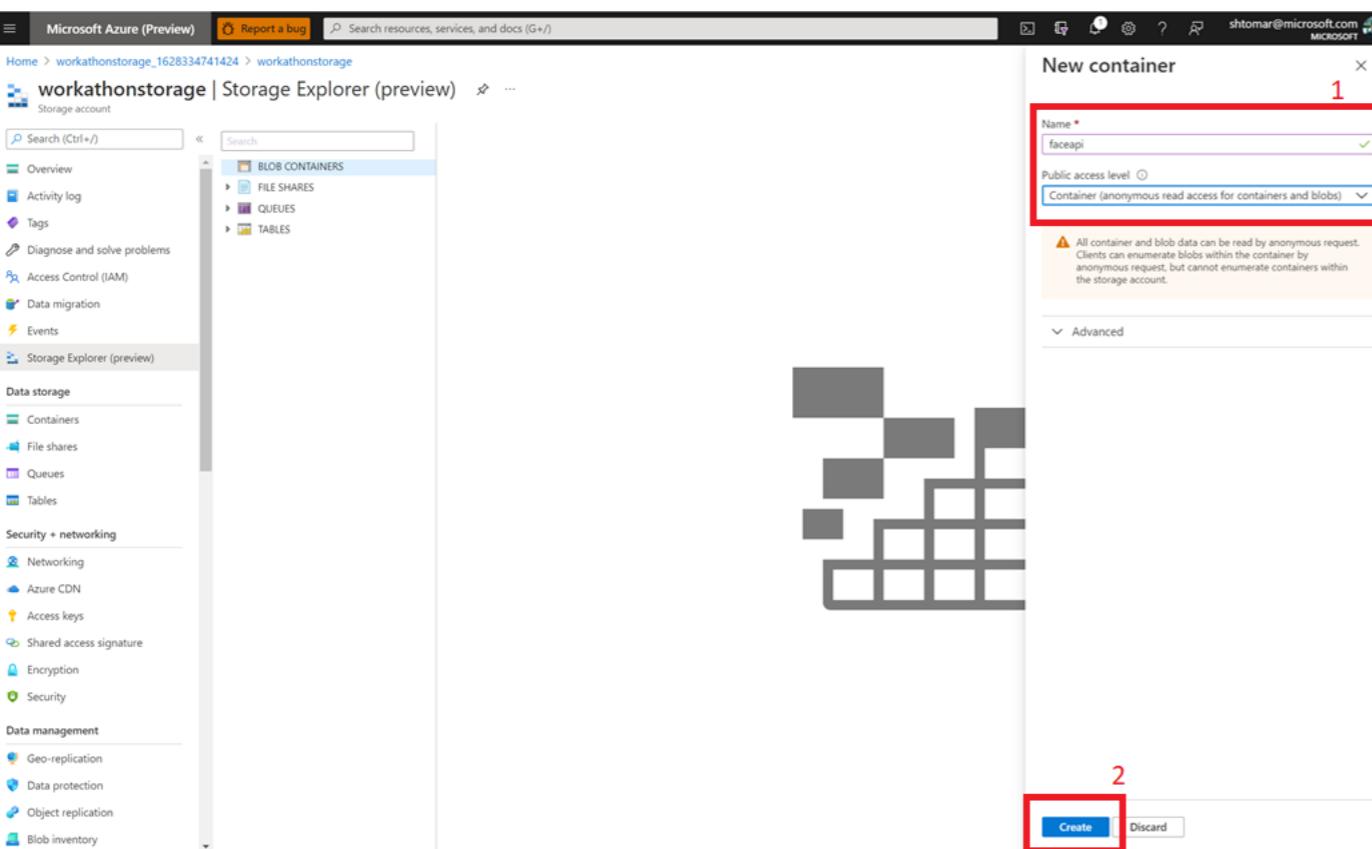


Verify the details and hit Create.



### Create Blob container

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



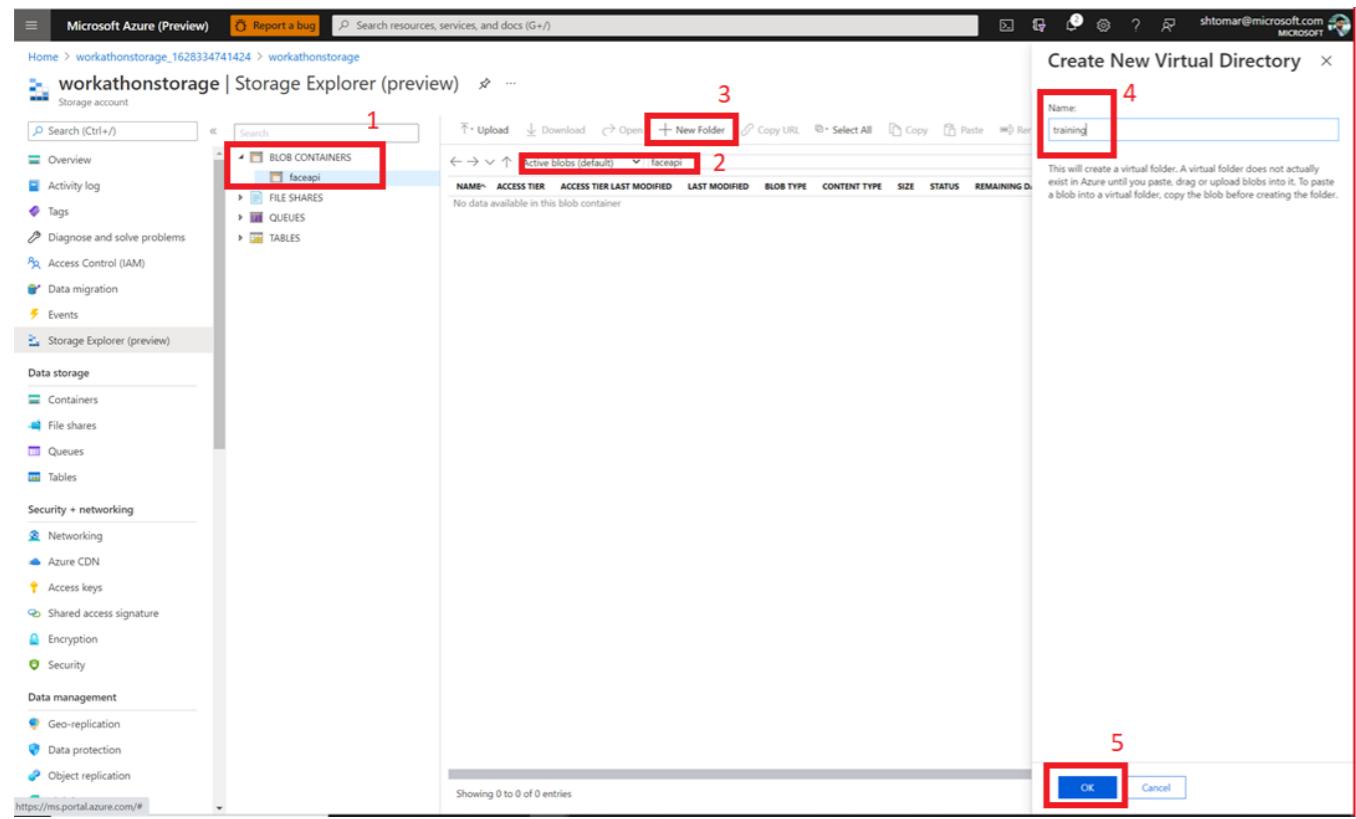
### Create a container with the following details –

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 for containers and blobs) \*\*

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.



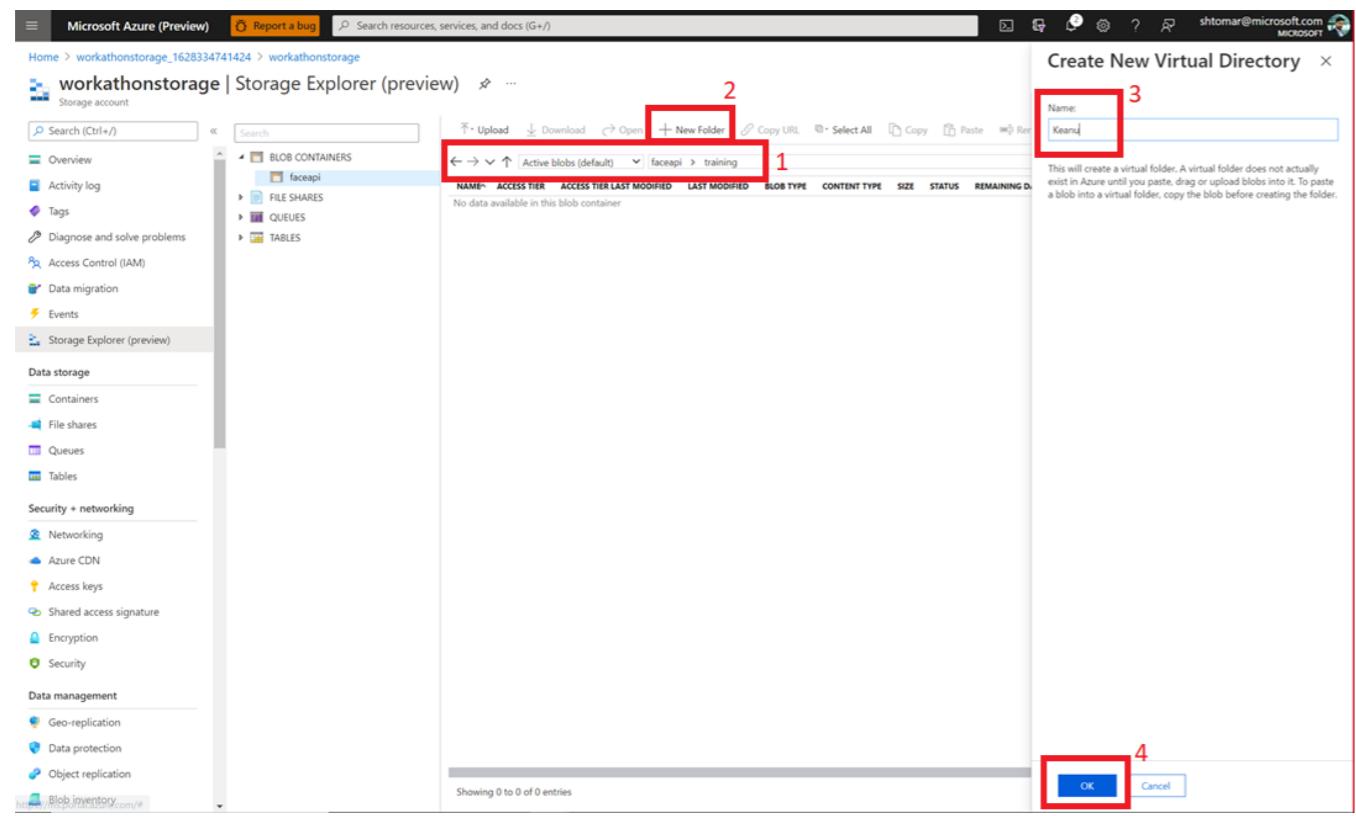
### Create folder for training data

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.



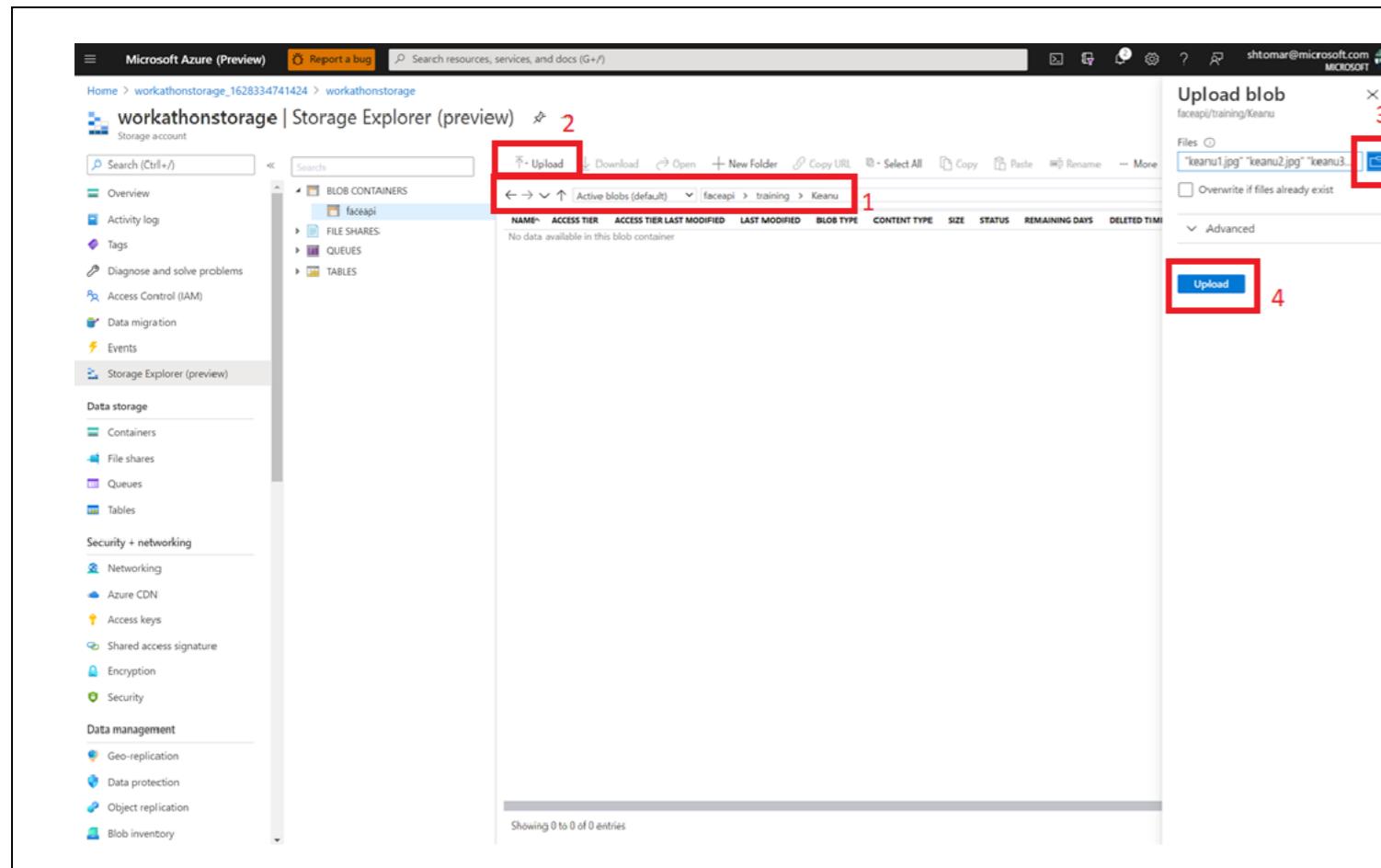
### Create folder for Person 1 (Keanu)

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.



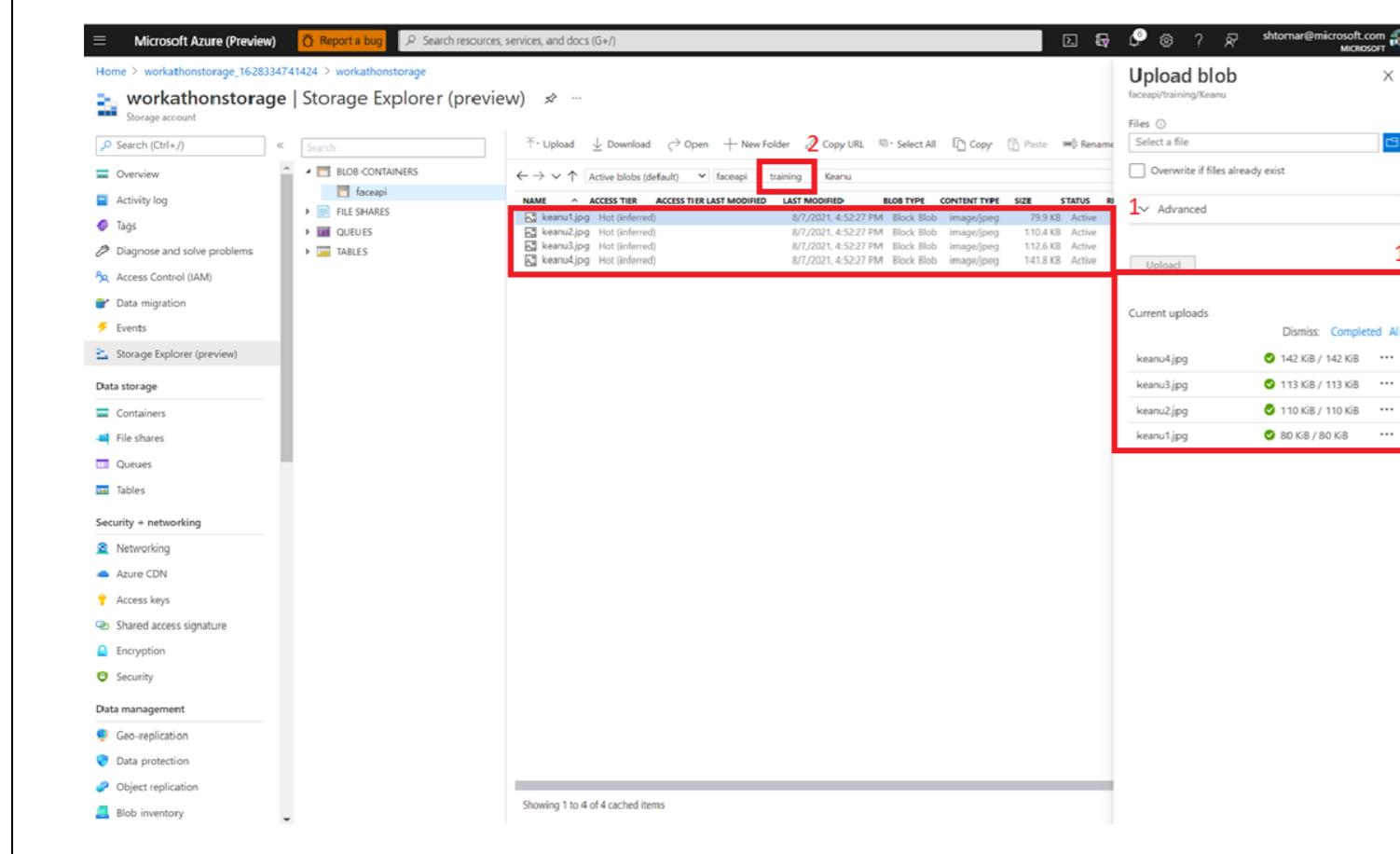
**Upload face images for person 1 (Keanu)**

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.



**Create folder & upload face images for Person 2 (scarjo)**

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.

The screenshot shows the Azure Storage Explorer interface. On the left, the navigation pane includes sections for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, and Storage Explorer (preview). The main area displays a list of blob containers under 'BLOB CONTAINERS'. One container, 'faceapi', is selected and expanded, showing its contents. A sub-container named 'test' is being created. A modal window titled 'Create New Virtual Directory' is open, with the 'Name:' field set to 'test'. Step numbers 1 through 4 are overlaid on the interface: 1 points to the 'faceapi' container in the list; 2 points to the '+ New Folder' button; 3 points to the 'Name:' input field; and 4 points to the 'OK' button at the bottom of the modal.

### Create folder for testing data

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.

The screenshot shows the Azure Storage Explorer interface with the 'faceapi/test' container selected. A modal window titled 'Upload blob' is open, showing a file selection dialog with 'Johanssontest.jpg' and 'keanutest.jpg' selected. Step numbers 1 through 4 are overlaid: 1 points to the 'faceapi/test' path in the address bar; 2 points to the 'Upload' button; 3 points to the file selection area; and 4 points to the 'Upload' button in the modal. Below the modal, a list of current uploads shows several image files (Johansson1.jpg, Johansson2.jpg, Johansson3.jpg, Johansson4.jpg, keanu1.jpg, keanu2.jpg, keanu3.jpg, keanu4.jpg) with their respective sizes and status.

### Upload test images

Make sure you are in the test folder, as shown in step 1.

Select Upload as shown in step 2.

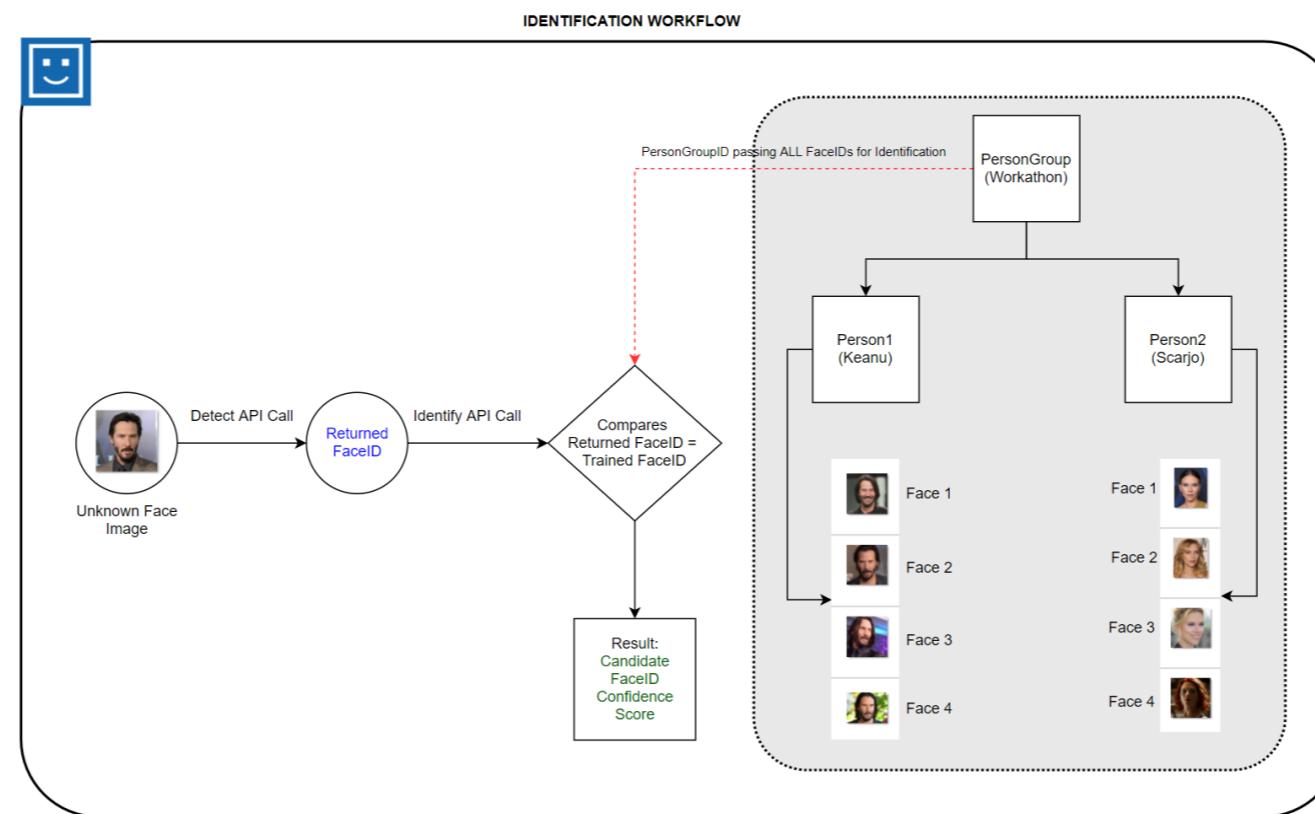
Click the file icon as shown in step 3 and browse & select the images you want to upload for testing your model.

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.



The screenshot shows the Postman interface. On the left, the sidebar has 'Collections' selected, showing 'ML-AI Workathon' and 'ComputerVision-OCR'. The main area shows the 'ComputerVision-OCR' collection details. A red box labeled '1' highlights the 'Add' button in the top right corner of the environment section. Another red box labeled '2' highlights the 'Add' button in the 'Globals' section. The bottom right corner of the interface shows standard Postman navigation buttons: 'Find and Replace', 'Console', 'Bootcamp', 'Runner', 'Trash', and a refresh icon.

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.

**Create new collection in Postman**

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.

The screenshot shows the Postman interface with the following steps highlighted:

- API Selection:** The "Create-PersonGroup" API under the "FaceAPI" collection is selected.
- Method:** The "PUT" method is chosen.
- URL:** The URL is set to {{endpoint}}/face/v1.0/persongroups/workathongrp.
- Headers:** The "Headers" tab is selected, showing two headers: "Ocp-Apim-Subscription-Key" with value {{key}} and "Content-Type" with value application/json.
- Body:** The "Body" tab is selected, showing the JSON body for creating the Person Group.

### 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.

The screenshot shows the Postman interface after executing the request, with the following steps highlighted:

6. The URL is again shown as {{endpoint}}/face/v1.0/persongroups/workathongrp.
7. The "Body" tab is selected, showing the JSON body.
8. The "Send" button is highlighted.
9. The "Cookies" tab is selected.
10. The response status is shown as Status: 200 OK.
11. The "Headers" tab is selected, showing the response headers.

KEY	VALUE
Content-Length	0
x-envoy-upstream-service-time	384
apim-request-id	a958586d-2c14-410c-83d6-baa257b65b12
Strict-Transport-Security	max-age=31536000; includeSubDomains; preload
x-content-type-options	nosniff
CSP-Billing-Usage	CognitiveServices.Face.Transaction=1
Date	Sat, 07 Aug 2021 11:33:59 GMT

The screenshot shows the Postman application interface. On the left, the sidebar lists 'ML-AI Workathon' collections, 'ComputerVision-ImageAnalysis', 'ComputerVision-OCR', and 'FaceAPI'. Under 'FaceAPI', there are three items: 'PUT Create-PersonGroup' (1), 'POST Create-Person1' (2), and 'POST Create-Person2' (3). The 'POST Create-Person1' request is selected. The top bar shows the URL: {{endpoint}}/face/v1.0/persongroups/workathongrp/persons. The 'Body' tab is selected (4), showing a JSON payload (5) with the following content:

```

1
2   "name": "keanu",
3   "userData": "Adding person 1 in the persongroup workathongrp"
4

```

The 'Send' button (7) is highlighted. The response panel (8) shows a status of '200 OK' with a response time of 1555 ms and a size of 428 B. The response body (9) contains:

```

1
2   "personId": "37f63b95-1485-4325-9451-497978a27eab"
3

```

### 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", "persistedFacelDs", "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. "persistedFacelDs" is an empty array since we haven't added any faces to the Person Objects yet. We will do that in the next steps.

The image contains two screenshots of the Postman application interface, demonstrating the process of adding faces to person objects.

**Screenshot 1 (Top):** Shows the 'Create-Face1' POST request configuration. 
 

- Step 1:** The 'POST Create-Face1' item is selected in the left sidebar.
- Step 2:** The 'POST' method is selected in the top bar.
- Step 3:** The URL is set to: `{endpoint}/face/v1.0/persongroups/workathongrp/persons/4bbec562-f0fb-4a8d-9c6c-580811a64c6d/persistedFaces?userData='person1 face1"&detectionModel=detection_03`.
- Step 4:** The Body tab is selected, showing the JSON body: `{"url": "https://workathonstorage.blob.core.windows.net/faceapi/training/Keanu/keanu1.jpg"}`.
- Step 5:** The 'Send' button is highlighted.

**Screenshot 2 (Bottom):** Shows the 'Create-Face1' POST request execution results.
 

- Step 6:** The 'POST Create-Face1' item is selected in the left sidebar.
- Step 7:** The 'Params' section shows parameters: `userData: "person1 face1"`, `detectionModel: detection_03`.
- Step 8:** The 'Send' button is highlighted.
- Step 9:** The status bar shows 'Status: 200 OK'.
- Step 10:** The response body is displayed as JSON: `[{"persistedFaceId": "d7484553-a0a6-40a0-8f6c-54037de7157b"}]`.

### 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 `persistedFacetIds` field returned in the result. You should see 4 IDs for each Person object.

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
[{"personId": "37f63b05-1485-4325-9451-497078a27eab",
 "persistedFacetIds": [
 "014c-c7f-8740-4038-81dc-7a8a345309a2",
 "04fc117d-059a-4203-89a2-cd95b6aa0a08",
 "cd5a601b-59fc-48db-adte-e8dd69f9fe6a",
 "de661948-c339-4927-ab6-5cd295d30817"
 ],
 "name": "scarjo",
 "userData": "Adding person 2 in the persongroup workathongrp"
},
{
"personId": "4bec562-f0fb-4a8d-9c6c-580811a64c6d",
"persistedFacetIds": [
"9fe009db-83b2-452e-80c8-08f9012e5e8a",
"3e92215-eeeb-4d96-945e-b4d310bbdd10",
"d7484c553-a0a6-40a0-8f6c-54037e71570",
"52f1edax-17c3-40bc-b719-b345151729"
],
"name": "keanu",
"userData": "Adding person 1 in the persongroup workathongrp"
}
]

```

The image contains two side-by-side screenshots of the Postman application interface.

**Screenshot 1: POST Train-PersonGroup**

- Left Panel:** Shows the 'ML-AI Workathon' collection with several APIs listed. The 'POST Train-PersonGroup' API is selected, highlighted with a red box labeled '1'.
- Request Tab:** A POST request to `{(endpoint)}/face/v1.0/persongroups/workathongrp/train`. The 'Headers' tab is selected, showing the following header:
 

KEY	VALUE
Ocp-Apim-Subscription-Key	<code>({key})</code>
Content-Type	application/json

 A red box labeled '2' highlights the endpoint URL. A red box labeled '3' highlights the 'Send' button. A red box labeled '4' highlights the 'Headers' tab. A red box labeled '5' highlights the 'Send' button again. A red box labeled '6' highlights the status bar at the bottom of the response panel.
- Response Tab:** Shows the response details. The status is 202 Accepted, time is 1879 ms, size is 507 B. The response body is a JSON object with one item:
 

```

1: {
  "status": "succeeded",
  "createdDateTime": "2021-08-07T12:38:55.182914Z",
  "lastActionDateTime": "2021-08-07T12:38:55.602369Z",
  "lastSuccessfulTrainingId": "a79fa3e4-062d-4a22-ab08-4e5a422e80e1",
  "lastSuccessfulTrainingDateTime": "2021-08-07T12:38:55.602369Z"
}
      
```

**Screenshot 2: GET Get-TrainingStatus**

- Left Panel:** Shows the same collection and 'GET Get-TrainingStatus' API selected, highlighted with a red box labeled '1'.
- Request Tab:** A GET request to `{(endpoint)}/face/v1.0/persongroups/workathongrp/training`. The 'Headers' tab is selected, showing the following header:
 

KEY	VALUE
Ocp-Apim-Subscription-Key	<code>({key})</code>
Content-Type	application/json
- Response Tab:** Shows the response details. The status is 200 OK, time is 382 ms, size is 631 B. The response body is a JSON object with one item:
 

```

1: {
  "status": "succeeded",
  "createdDateTime": "2021-08-07T12:38:55.182914Z",
  "lastActionDateTime": "2021-08-07T12:38:55.602369Z",
  "lastSuccessfulTrainingId": "a79fa3e4-062d-4a22-ab08-4e5a422e80e1",
  "lastSuccessfulTrainingDateTime": "2021-08-07T12:38:55.602369Z"
}
      
```

## 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.

The image consists of two vertically stacked screenshots of the Postman application interface.

**Screenshot 1 (Top):** Shows the 'ML-AI Workathon' collection. A red box highlights the 'Detect-Face' item under the 'FaceAPI' section. The URL field contains `({{endpoint}})/face/v1.0/detect?returnFacelId=true&returnFaceLandmarks=true&recognitionModel=recognition_04&returnRecognitionModel=true&detectionModel=detection_03`. The 'Params' tab shows several checked parameters: returnFacelId, returnFaceLandmarks, recognitionModel (set to 'recognition\_04'), detectionModel (set to 'detection\_03'), and facelTimeToLive (set to 86400). The 'Send' button is highlighted with a red box.

**Screenshot 2 (Bottom):** Shows the same 'ML-AI Workathon' collection. The 'Body' tab is selected, containing the JSON body: `{"url": "https://workathonstorage.blob.core.windows.net/faceapi/test/Johanssontest.jpg"}`. The 'Send' button is again highlighted with a red box. The response panel shows a successful 200 OK status with a JSON payload containing face detection details like bounding boxes and landmarks.

## 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 **facelId** 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 **facelId** 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:

returnFacelId : true  
 returnFaceLandmarks : true  
 recognitionModel : recognition\_04  
 detectionModel : detection\_03  
 facelTimeToLive : 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}}
```

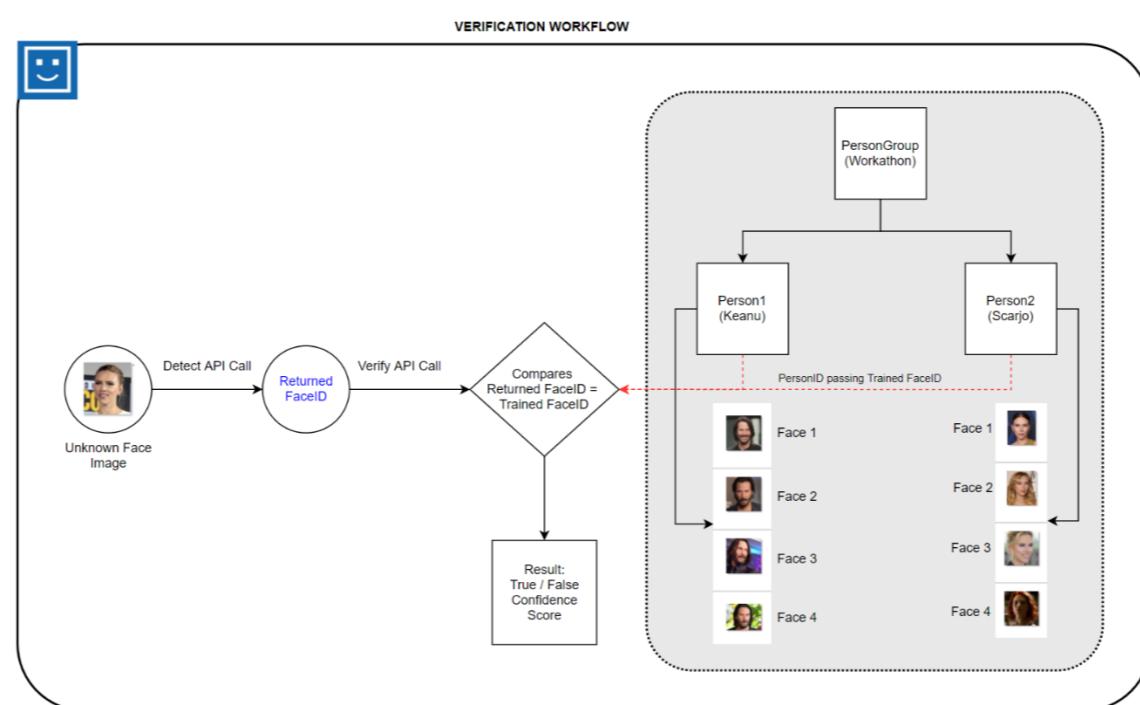
Content-Type : application/json

Body :

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

#### 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 :

```
{
  "facelid": "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. **facelid** is the unique ID of input faces you want to verify. These facelid 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   ( <a href="#">Differences between Tiers</a> )
	Pricing	<a href="https://azure.microsoft.com/en-in/pricing/details/cognitive-services/face-api/">https://azure.microsoft.com/en-in/pricing/details/cognitive-services/face-api/</a>
	Limits	
	Language Support	Not applicable
	Sample Apps	<a href="#">FamilyNotes</a> , <a href="#">ReactApp</a>
	Regional Availability & Support	<a href="https://azure.microsoft.com/en-us/global-infrastructure/services/?products=cognitive-services&amp;regions=all">https://azure.microsoft.com/en-us/global-infrastructure/services/?products=cognitive-services&amp;regions=all</a>
	SLAs for Cognitive Services	<a href="https://azure.microsoft.com/en-in/support/legal/sla/cognitive-services/v1_1/">https://azure.microsoft.com/en-in/support/legal/sla/cognitive-services/v1_1/</a>
	Compliance & Certificates	<a href="https://azure.microsoft.com/en-us/support/legal/cognitive-services-compliance-and-privacy/">https://azure.microsoft.com/en-us/support/legal/cognitive-services-compliance-and-privacy/</a>
	Cognitive Services Updates	<a href="https://azure.microsoft.com/en-us/updates/?product=cognitive-services">https://azure.microsoft.com/en-us/updates/?product=cognitive-services</a>

## 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](#)