Detect Labels, Faces, and Landmarks in Images with the Cloud Vision API

GSP037

Overview

The Cloud Vision API lets you understand the content of an image by encapsulating powerful machine learning models in a simple REST API.

In this lab, we will send images to the Vision API and see it detect objects, faces, and landmarks.

What you'll learn

- Creating a Vision API request and calling the API with `curl`
- Using the label, face, and landmark detection methods of the vision API

What you'll need

- A Google Cloud Project
- A Browser, such Chrome or Firefox

Setup and Requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

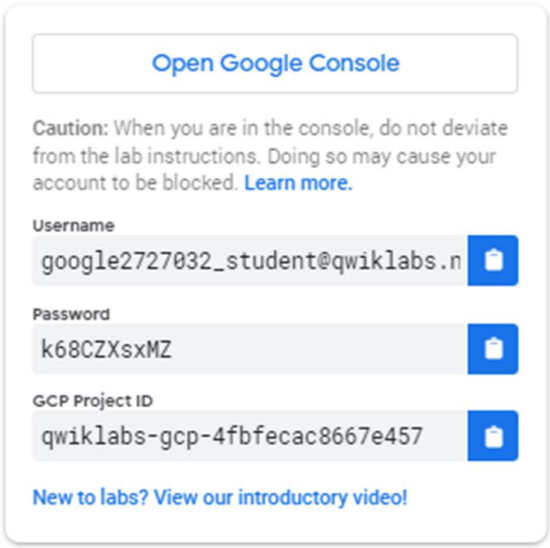
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

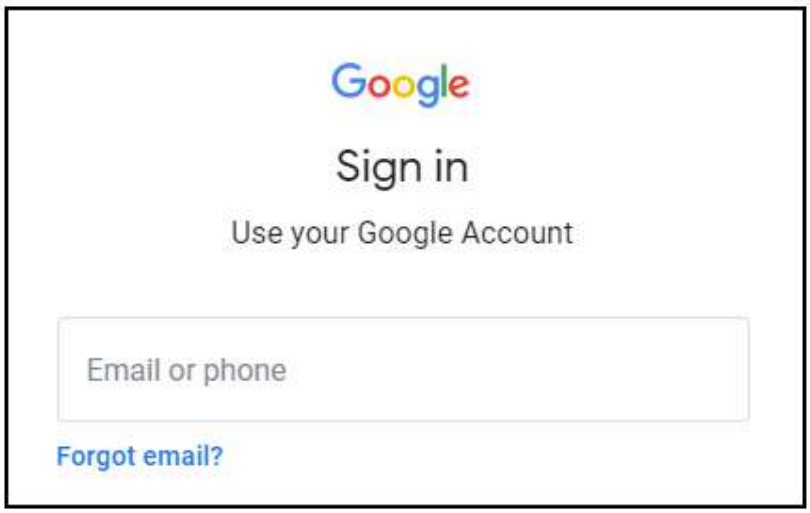
Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

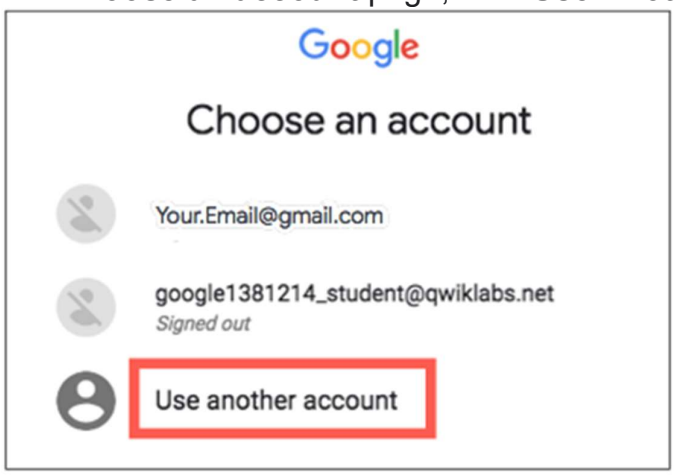


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

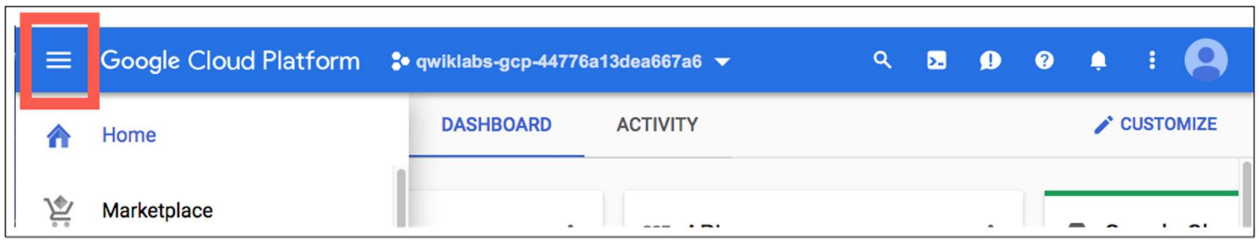
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

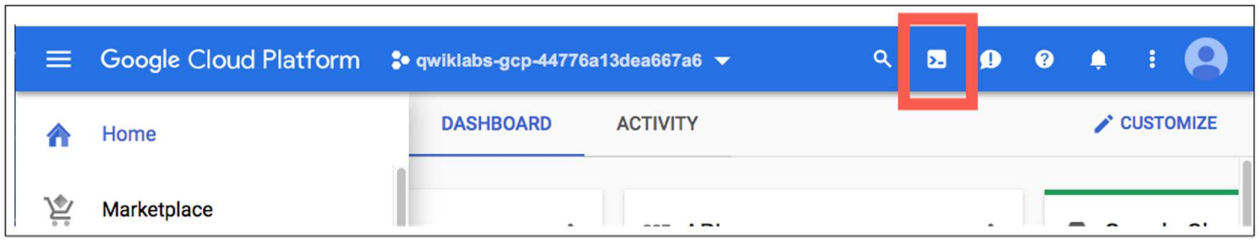
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



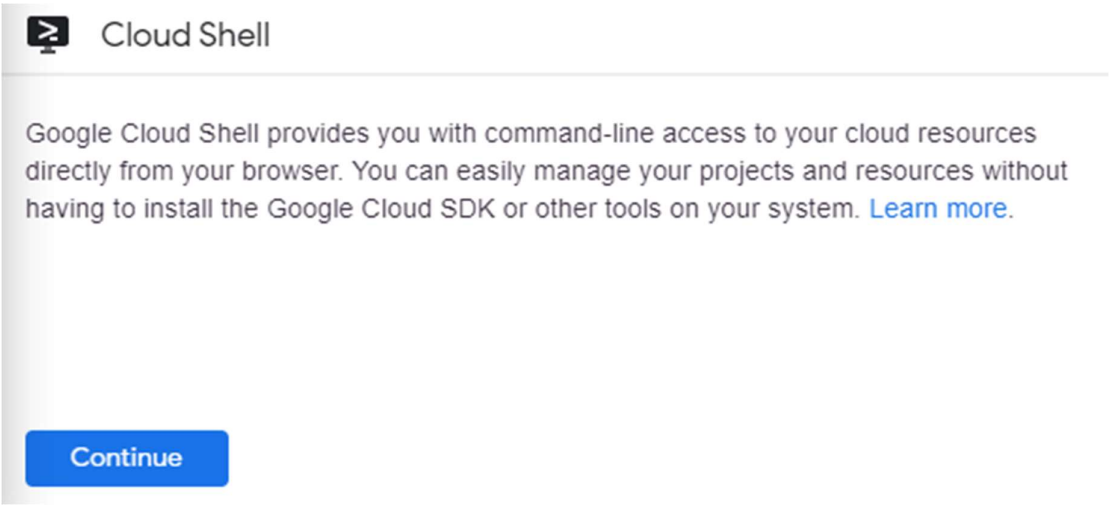
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]
project = <project_ID>
```

(Example output)

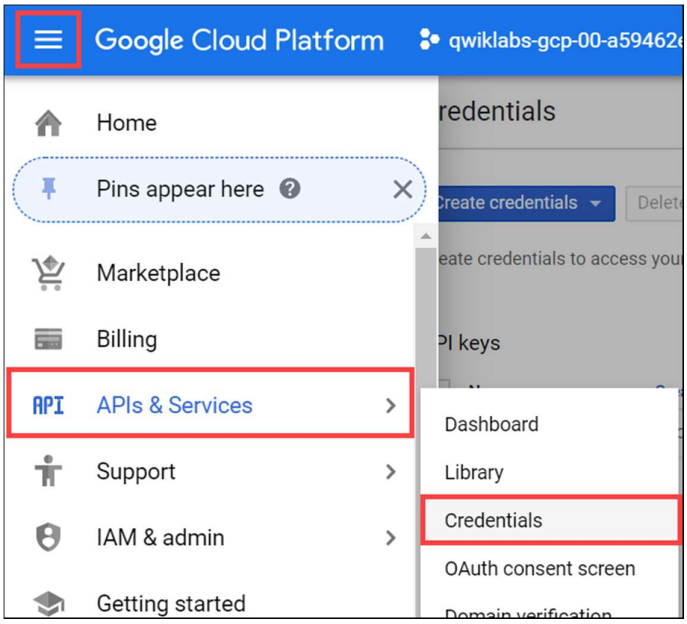
```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

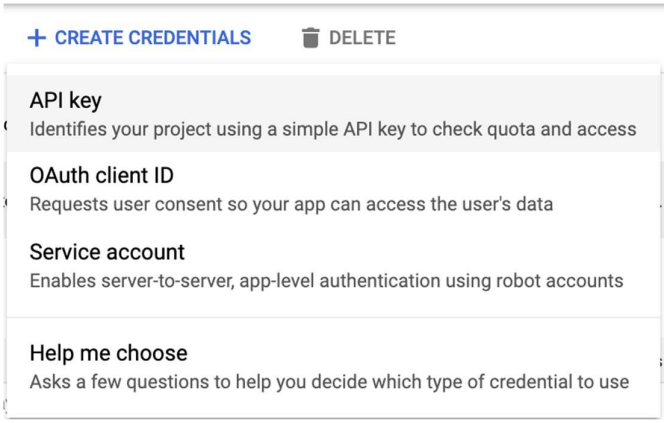
Create an API Key

Since you'll be using `curl` to send a request to the Vision API, you'll need to generate an API key to pass in your request URL.

1. To create an API key, navigate to **APIs & Services** > **Credentials** in your Cloud console:



2. Click **Create credentials** and select **API key**.



3. Next, copy the key you just generated and click **Close**.

Click **Check my progress** below to check your lab progress.

Create an API Key

Check my progress

Now save it to an environment variable to avoid having to insert the value of your API key in each request.

Run the following in Cloud Shell, replacing `<your_api_key>` with the key you just copied:

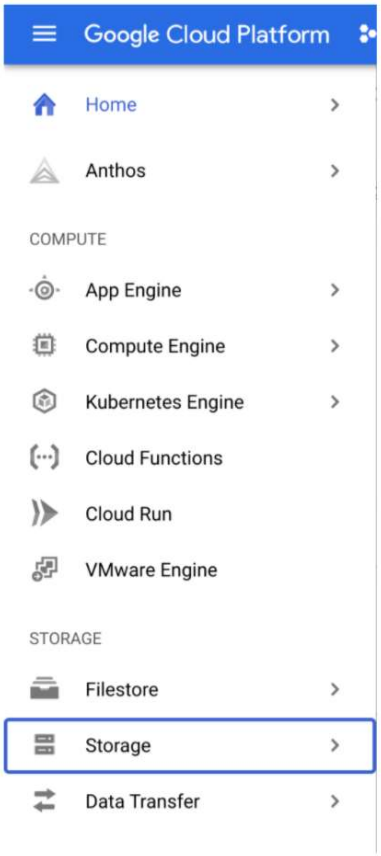
```
export API_KEY=<YOUR_API_KEY>
```

Upload an Image to a Cloud Storage bucket

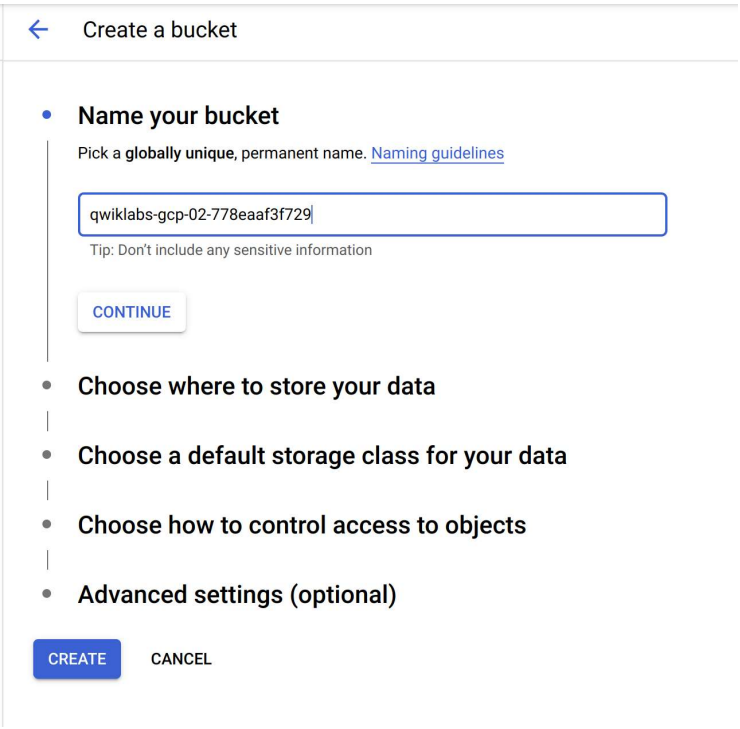
Creating a Cloud Storage bucket

There are two ways to send an image to the Vision API for image detection: by sending the API a base64 encoded image string, or passing it the URL of a file stored in Cloud Storage. We'll be using a Cloud Storage URL. The first step is to create a Cloud Storage bucket to store our images.

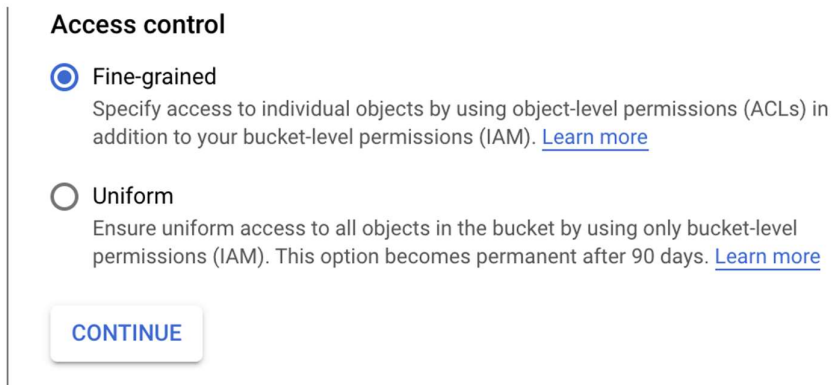
1. Navigate to **Navigation menu** > **Storage** in the Cloud console for your project, then click **Create bucket**.



2. Give your bucket a unique name.



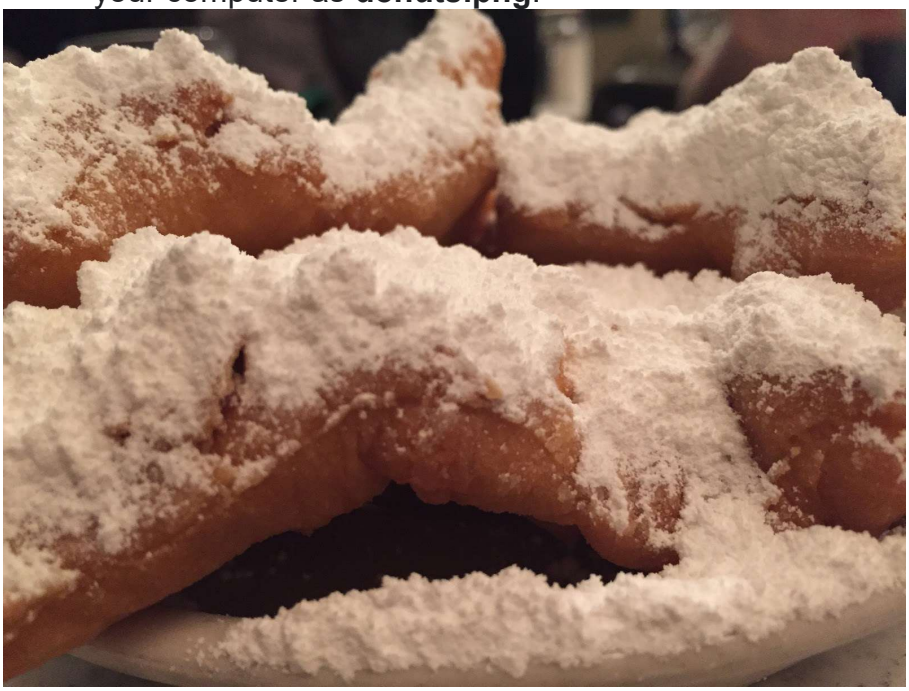
3. After naming your bucket, click **Choose how to control access to objects** and select the **Fine-grained** circle:



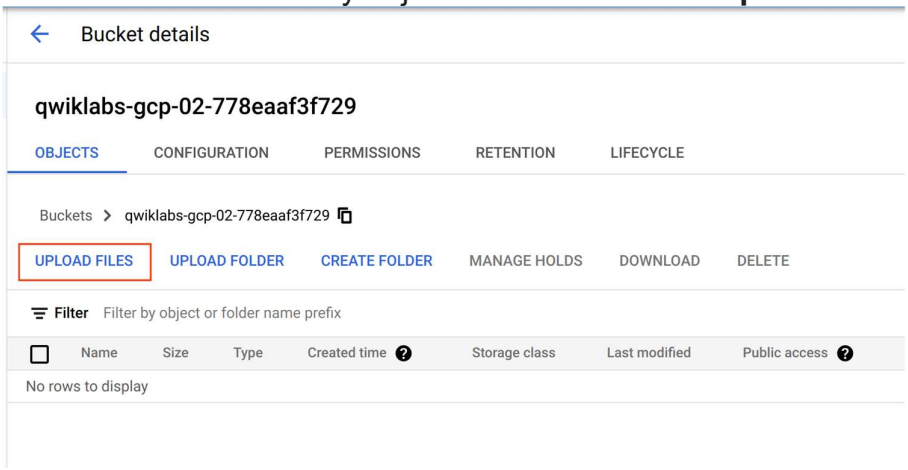
4. All other settings for your bucket can remain as the default setting. Click **Create**.

Upload an image to your bucket

1. Right click on the following image of donuts, then click **Save image as** and save it to your computer as **donuts.png**.

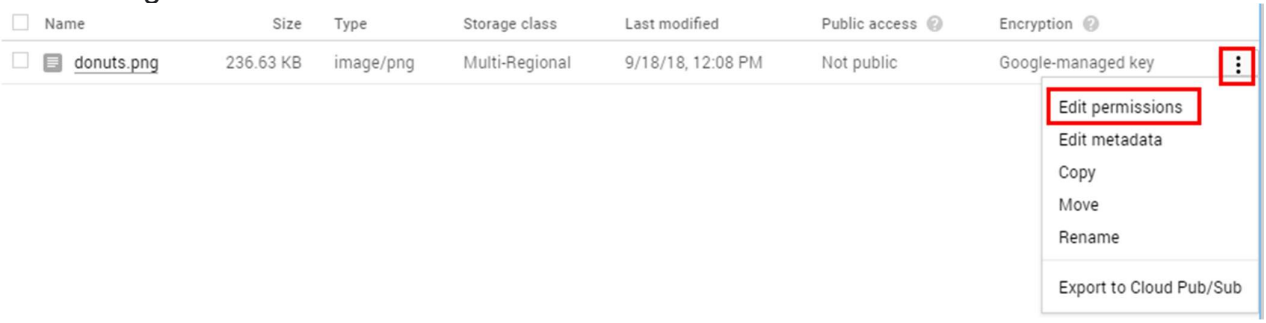


2. Go to the bucket you just created and click **Upload files**. Then select **donuts.png**.



You should see the file in your bucket.

3. Now you need to make this image publicly available. Click on the 3 dots for your image and select **Edit Permissions**.

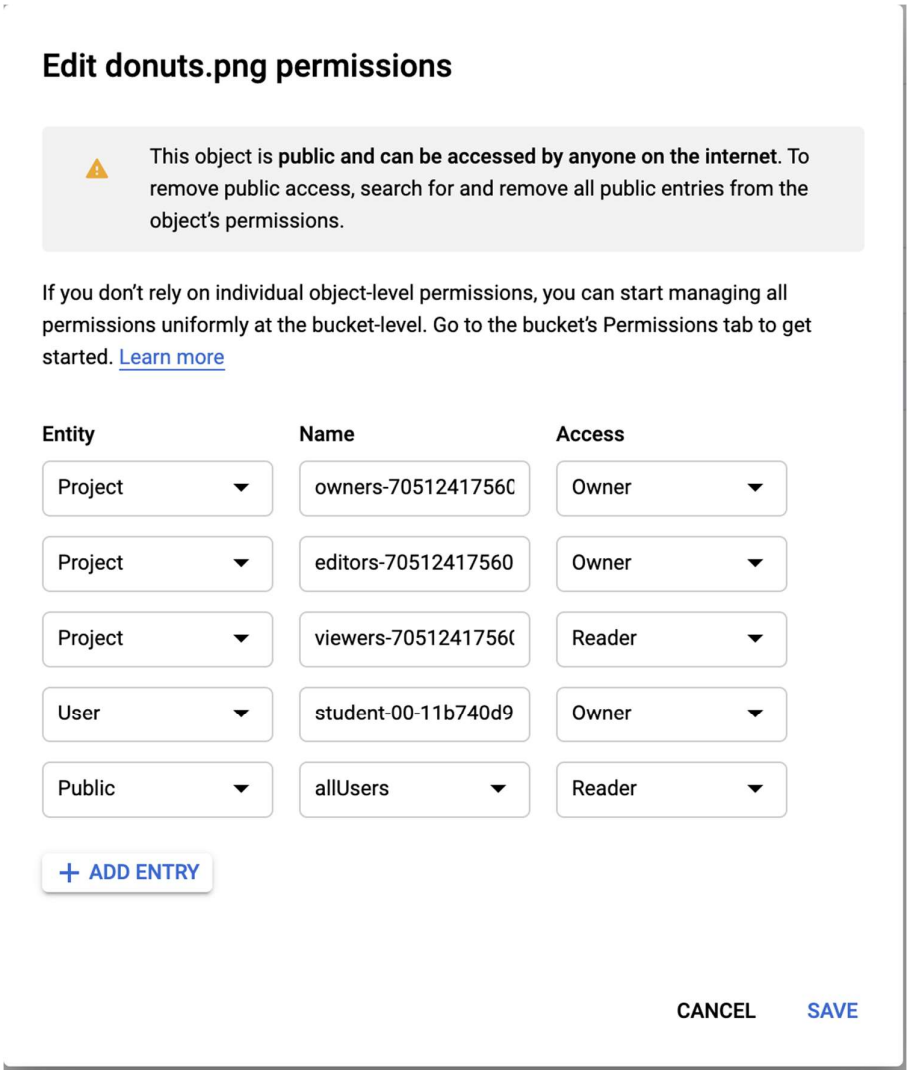


4. Click **Add entry** then enter the following:

Entity: Public

Name: allUsers

Access: Reader



5. Then click **Save**.

Now that you have the file in your bucket, you're ready to create a Vision API request, passing it the URL of this donuts picture.

Click **Check my progress** below to check your lab progress.

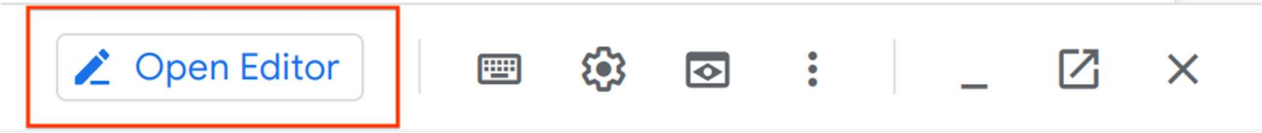
Upload an image to your bucket

Check my progress

Create your Vision API request

Now you'll create a `request.json` file in the Cloud Shell environment.

1. Using the Cloud Shell code editor (by clicking the pencil icon in the Cloud Shell ribbon),



or your preferred command line editor (`nano`, `vim`, or `emacs`), create a `request.json` file.

2. Type or paste the following code into the file:

Note: Replace `my-bucket-name` with the name of your storage bucket.

```
{
  "requests": [
      {
        "image": {
          "source": {
              "gcsImageUri": "gs://my-bucket-name/donuts.png"
          }
        },
        "features": [
          {
            "type": "LABEL_DETECTION",
            "maxResults": 10
          }
        ]
      }
  ]
}
```

3. **Save** the file.

Label Detection

The first Cloud Vision API feature you'll try out is label detection. This method will return a list of labels (words) of what's in your image.

Call the Vision API with `curl`:

```
curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json
https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

Your response should look something like the following:


```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/01dk8s",
          "description": "Powdered sugar",
          "score": 0.9861496,
          "topicality": 0.9861496
        },
        {
          "mid": "/m/01wydv",
          "description": "Beignet",
          "score": 0.9565117,
          "topicality": 0.9565117
        },
        {
          "mid": "/m/02wbm",
          "description": "Food",
          "score": 0.9424965,
          "topicality": 0.9424965
        },
        {
          "mid": "/m/0hnyx",
          "description": "Pastry",
          "score": 0.8173416,
          "topicality": 0.8173416
        },
        {
          "mid": "/m/02q08p0",
          "description": "Dish",
          "score": 0.8076026,
          "topicality": 0.8076026
        },
        {
          "mid": "/m/01ykh",
          "description": "Cuisine",
          "score": 0.79036003,
          "topicality": 0.79036003
        },
        {
          "mid": "/m/03nsjgy",
          "description": "Kourabiedes",
          "score": 0.77726763,
          "topicality": 0.77726763
        },
        {
          "mid": "/m/06gd3r",
          "description": "Angel wings",
          "score": 0.73792106,
```

```
          "topicality": 0.73792106
        },
        {
          "mid": "/m/06x4c",
          "description": "Sugar",
          "score": 0.71921736,
          "topicality": 0.71921736
        },
        {
          "mid": "/m/01zl9v",
          "description": "Zeppole",
          "score": 0.7111677,
          "topicality": 0.7111677
        }
      ]
    }
  ]
}
```


The API was able to identify the specific type of donuts these are, powdered sugar. Cool! For each label the Vision API found, it returns a:

- `description` with the name of the item.
- `score`, a number from 0 - 1 indicating how confident it is that the description matches what's in the image.
- `mid` value that maps to the item's `mid` in Google's Knowledge Graph. You can use the `mid` when calling the Knowledge Graph API to get more information on the item.

Web Detection

In addition to getting labels on what's in your image, the Vision API can also search the Internet for additional details on your image. Through the API's webDetection method, you get a lot of interesting data back:

- A list of entities found in your image, based on content from pages with similar images
- URLs of exact and partial matching images found across the web, along with the URLs of those pages
- URLs of similar images, like doing a reverse image search

To try out web detection, use the same image of beignets and change one line in the `request.json` file (you can also venture out into the unknown and use an entirely different image).

1. Under the features list, change **type** from `LABEL_DETECTION` to `WEB_DETECTION`. The `request.json` should now look like this:

```
{
  "requests": [
      {
        "image": {
          "source": {
              "gcsImageUri": "gs://my-bucket-name/donuts.png"
          }
        },
        "features": [
          {
            "type": "WEB_DETECTION",
            "maxResults": 10
          }
        ]
      }
  ]
}
```

Save the file.

2. To send it to the Vision API, use the same `curl` command as before (just press the up arrow in Cloud Shell):

```
curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json
https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

Dive into the response, starting with `webEntities`. Here are some of the entities this image returned:

```
{
  "responses": [
    {
      "webDetection": {
        "webEntities": [
          {
            "entityId": "/m/0z5n",
            "score": 0.8868,
            "description": "Application programming interface"
          },
          {
            "entityId": "/m/07kg1sq",
            "score": 0.3139,
            "description": "Encapsulation"
          },
          {
            "entityId": "/m/0105pbj4",
            "score": 0.2713,
            "description": "Google Cloud Platform"
          },
          {
            "entityId": "/m/01hyh_",
            "score": 0.2594,
            "description": "Machine learning"
          },
          ...
        ]
```


This image has been used in many presentations on Cloud ML APIs, which is why the API found the entities "Machine learning" and "Google Cloud Platform".

If you inpsect the URLs under `fullMatchingImages`, `partialMatchingImages`, and `pagesWithMatchingImages`, you'll notice that many of the URLs point to this lab site (super meta!).

Say you wanted to find other images of beignets, but not the exact same images. That's where the `visuallySimilarImages` part of the API response comes in handy. Here are a few of the visually similar images it found:


```
        "visuallySimilarImages": [
          {
            "url": "https://media.istockphoto.com/photos/cafe-du-monde-picture-
id1063530570?k=6&m=1063530570&s=612x612&w=0&h=b74EYAjlfxMw8G-G_6BW-6ltP9Y2UFQ3TjZopN-
pigI="
          },
          {
            "url": "https://s3-
media2.fl.yelpcdn.com/bphoto/oid0KchdCqlSqZzpznCEoA/o.jpg"
          },
          {
            "url": "https://s3-
media1.fl.yelpcdn.com/bphoto/mgAhrlLFvXe0IkT5UMOUlw/348s.jpg"
          },

          ...
]
```


You can navigate to those URLs to see the similar images:

And now you probably really want a powdered sugar beignet (sorry)! This is similar to searching by an image on Google Images.
With Cloud Vision you can access this functionality with an easy to use REST API and integrate it into your applications.

Face Detection

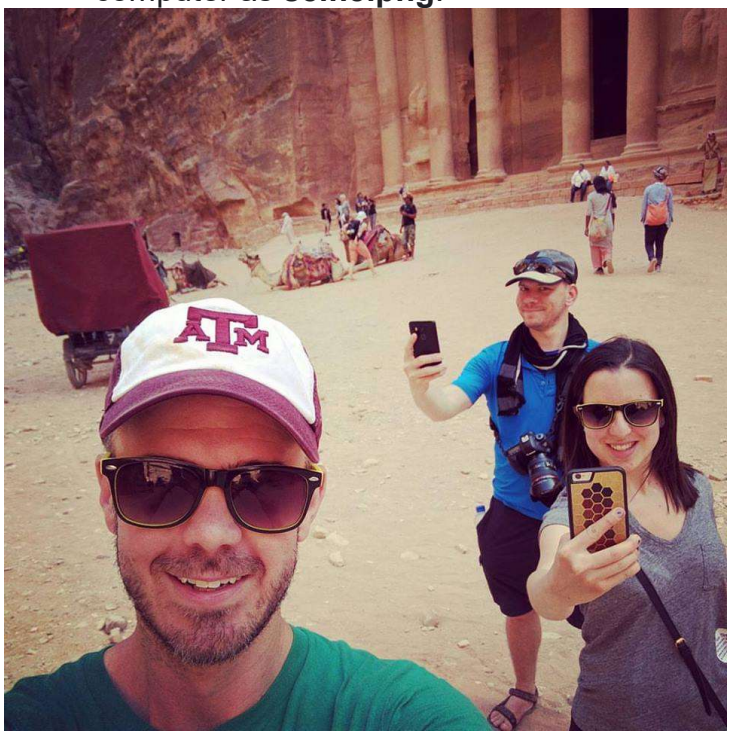
Next explore the face detection methods of the Vision API.

- The face detection method returns data on faces found in an image, including the emotions of the faces and their location in the image.

Upload a new image

To use this method, you'll upload a new image with faces to the Cloud Storage bucket.

1. Right click on the following image, then click **Save image as** and save it to your computer as **selfie.png**.



2. Now upload it to your Cloud Storage bucket the same way you did before, and make it public.

Click **Check my progress** below to check your lab progress.

Updating request file

1. Next, update your `request.json` file with the following, which includes the URL of the new image, and uses face and landmark detection instead of label detection. Be sure to replace **my-bucket-name** with the name of your Cloud Storage bucket:

```
{
  "requests": [
      {
        "image": {
          "source": {
              "gcsImageUri": "gs://my-bucket-name/selfie.png"
          }
        },
        "features": [
          {
            "type": "FACE_DETECTION"
          },
          {
            "type": "LANDMARK_DETECTION"
          }
        ]
      }
  ]
}
```

2. **Save** the file.

Calling the Vision API and parsing the response

Now you're ready to call the Vision API using the same `curl` command you used above:

```
curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json
https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

Take a look at the `faceAnnotations` object in the response. You'll notice the API returns an object for each face found in the image - in this case, three. Here's a clipped version of the response:

```
{
      "faceAnnotations": [
        {
          "boundingPoly": {
            "vertices": [
              {
                "x": 669,
                "y": 324
              },
              ...
            ]
          },
          "fdBoundingPoly": {
            ...
          },
          "landmarks": [
            {
              "type": "LEFT_EYE",
              "position": {
                "x": 692.05646,
                "y": 372.95868,
                "z": -0.00025268539
              }
            },
            ...
          ],
          "rollAngle": 0.21619819,
          "panAngle": -23.027969,
          "tiltAngle": -1.5531756,
          "detectionConfidence": 0.72354823,
          "landmarkingConfidence": 0.20047489,
          "joyLikelihood": "LIKELY",
          "sorrowLikelihood": "VERY_UNLIKELY",
          "angerLikelihood": "VERY_UNLIKELY",
          "surpriseLikelihood": "VERY_UNLIKELY",
          "underExposedLikelihood": "VERY_UNLIKELY",
          "blurredLikelihood": "VERY_UNLIKELY",
          "headwearLikelihood": "VERY_LIKELY"
        }
        ...
      }
}
```


- `boundingPoly` gives you the x,y coordinates around the face in the image.
- `fdBoundingPoly` is a smaller box than `boundingPoly`, focusing on the skin part of the face.
- `landmarks` is an array of objects for each facial feature, some you may not have even known about. This tells us the type of landmark, along with the 3D position of that feature (x,y,z coordinates) where the z coordinate is the depth. The remaining values gives you more details on the face, including the likelihood of joy, sorrow, anger, and surprise.
  The response you're reading is for the person standing furthest back in the image - you can see he's making kind of a silly face which explains the `joyLikelihood` of `LIKELY`.

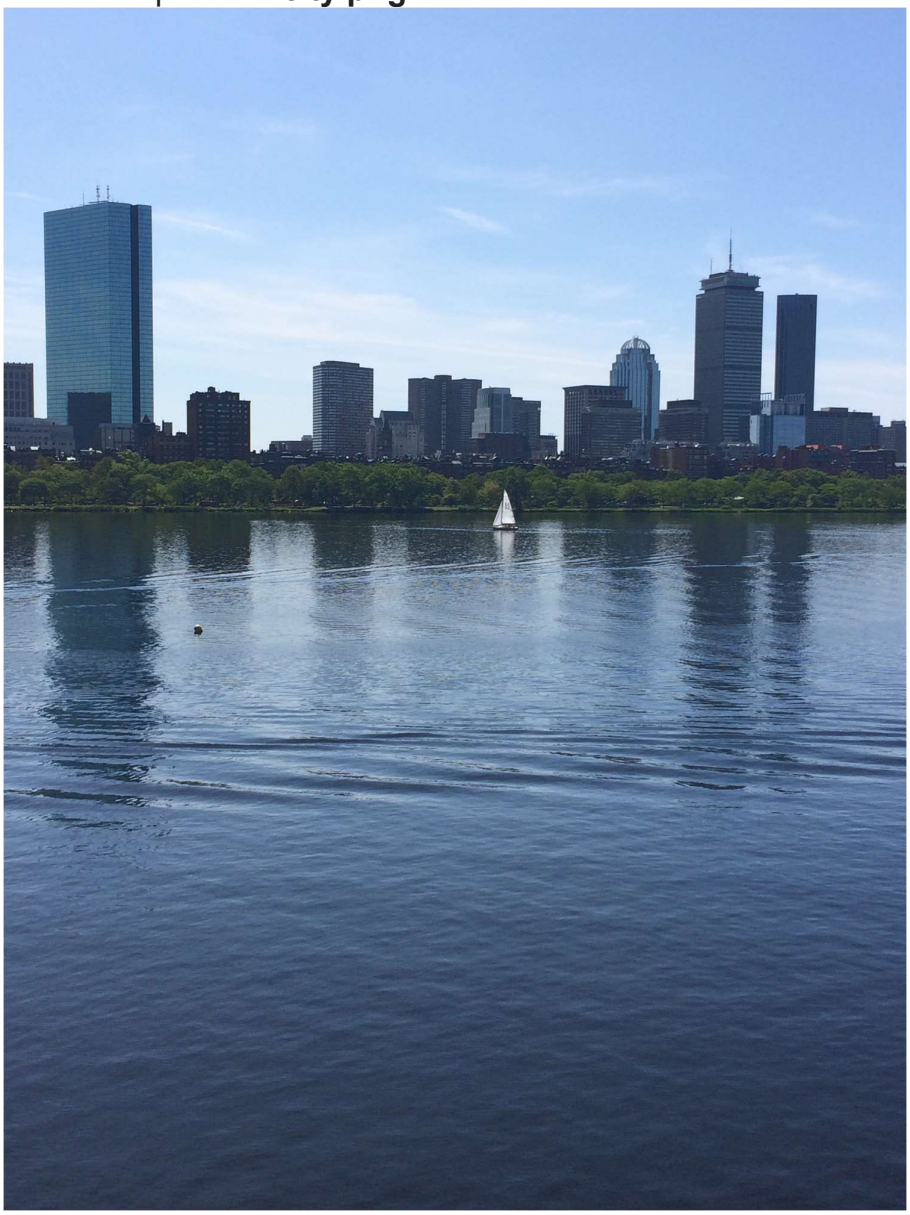
Landmark Annotation

- Landmark detection can identify common (and obscure) landmarks. It returns the name of the landmark, its latitude and longitude coordinates, and the location of where the landmark was identified in an image.

Upload a new image

To use this method, you'll upload a new image with faces to the Cloud Storage bucket.

1. Right click on the following image, then click **Save image as** and save it to your computer as **city.png**.



2. Now upload it to your Cloud Storage bucket the same way you did before, and make it public.

Click **Check my progress** below to check your lab progress.

Upload an image for Landmark Annotation to your bucket

Check my progress

Updating request file

1. Next, update your `request.json` file with the following, which includes the URL of the new image, and uses landmark detection. Be sure to replace **my-bucket-name** with the name of your Cloud Storage bucket:

```
{
  "requests": [
      {
        "image": {
          "source": {
              "gcsImageUri": "gs://my-bucket-name/city.png"
          }
        },
        "features": [
          {
            "type": "LANDMARK_DETECTION",
            "maxResults": 10,
          }
        ]
      }
  ]
}
```

Calling the Vision API and parsing the response

Now you're ready to call the Vision API using the same `curl` command you used above:

```
curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json
https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

Next, look at the `landmarkAnnotations` part of the response:

```
      "landmarkAnnotations": [
        {
          "mid": "/m/041cp3",
          "description": "Boston",
          "score": 0.788803,
          "boundingPoly": {
            "vertices": [
              {
                "y": 576
              },
              {
                "x": 1942,
                "y": 576
              },
              {
                "x": 1942,
                "y": 1224
              },
              {
                "y": 1224
              }
            ]
          },

...
```

Here, the Vision API was able to tell that this picture was taken in Boston, and gives you a map of the exact location. The values in this response should look similar to the `labelAnnotations` response above:

- the `mid` of the landmark
- it's name (`description`)
- a confidence `score`
- The `boundingPoly` shows the region in the image where the landmark was identified.
- The `locations` key tells us the latitude longitude coordinates of the picture.

Explore other Vision API methods

You've looked at the Vision API's label, face, and landmark detection methods, but there are three others you haven't explored. Dive into the docs to learn about the other three:

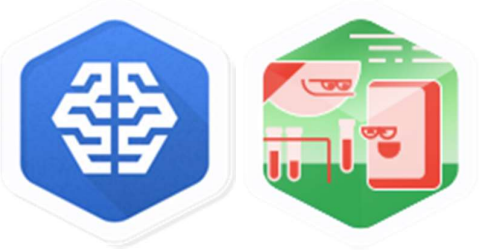
- **Logo detection**: identify common logos and their location in an image.
- **Safe search detection**: determine whether or not an image contains explicit content. This is useful for any application with user-generated content. You can filter images based on four factors: adult, medical, violent, and spoof content.
- **Text detection**: run OCR to extract text from images. This method can even identify the language of text present in an image.

Congratulations!

You've learned how to analyze images with the Vision API. In this example you passed the API the Cloud Storage URL of your image. Alternatively, you can pass a base64 encoded string of your image.

What you've covered

- Calling the Vision API with curl by passing it the URL of an image in a Cloud Storage bucket
- Using the Vision API's label, face, and landmark detection methods



Finish your quest

This self-paced lab is part of the Qwiklabs Quests Machine Learning APIs and Intro to ML: Image Processing. A Quest is a series of related labs that form a learning path. Completing a Quest earns you a badge to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in these Quests and get immediate completion credit if you've taken this lab. See other available Qwiklabs Quests.

Take your next lab

Try out another lab on Machine Learning APIs, like Entity and Sentiment Analysis with the Natural Language API or Awwvision: Cloud Vision API from a Kubernetes Cluster.

Next steps / learn more

- Check out the Vision API tutorials in the documentation
- Find a Vision API sample in your favorite language on GitHub
- Check out the Entity and Sentiment Analysis with the Natural Language API lab.

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.
Manual Last Updated March 17, 2021
Lab Last Tested February 04, 2021