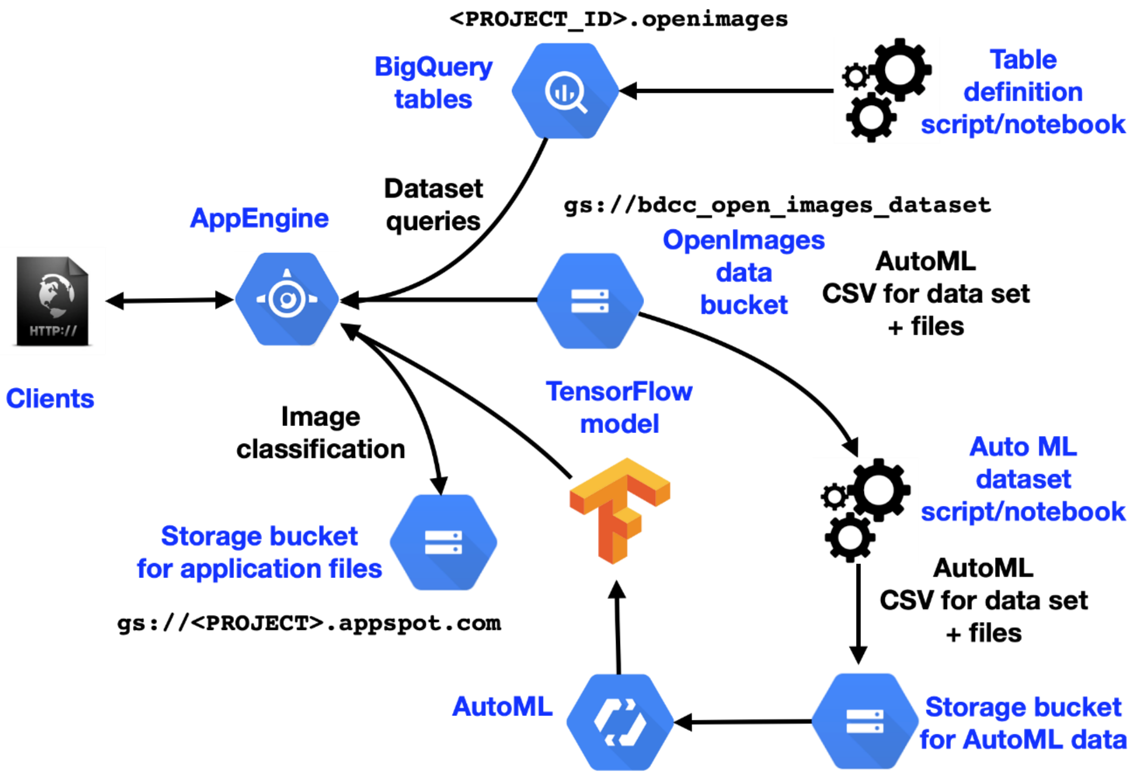
BDCC PROJECT

Gcloud ID: project-bdcc-nuno

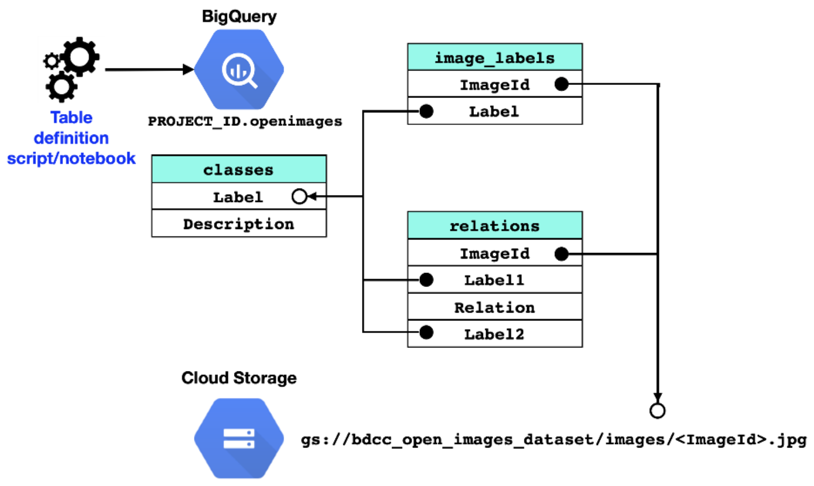
Nuno Fernandes (up202109069)

The goal of the present project was to deploy an app on Google Cloud using AppEngine following the demo application (https://bdcc22project.appspot.com). The app enables users to search over a dataset of images (i.e. by image Id) using BIgQuery and displays information about the categories of classes (i.e. people, animals, toys) and the relationship between these categories (i.e. human plays violin). Also, there were two additional services provided using the AI Cloud Vision API: image classification and labels.

The link to the app is https://project-bdcc-nuno.oa.r.appspot.com/ and an overview of the App functioning is schematized in the figure below.


<PROJECT_ID>.openimages
BigQuery tables
Table definition script/notebook
Dataset queries
AppEngine
gs://bdcc_open_images_dataset
OpenImages data bucket
AutoML CSV for data set + files
Clients
Image classification
TensorFlow model
Auto ML dataset script/notebook
Storage bucket for application files
gs://<PROJECT>.appspot.com
AutoML CSV for data set + files
AutoML
Storage bucket for AutoML data

BigQuery



After creating the *project-bdcc-nuno* project, the first step was to create an *openimages* data bucket to store *classes, image_labels, and relations* dataframes.

```
dataset = client.create_dataset('openimages', exists_ok=True)
```

Then, *classes, image_labels, and relations* BigQuery corresponding tables were created. Example for the *classes* dataframe:

```
table_name = PROJECT_ID + '.openimages.classes'
print('Creating table ' + table_name)

# Delete the table in case you're running this for the second time
client.delete_table(table_name, not_found_ok=True)

# Create the table
# - we use the same field names as in the original data set
table = bq.Table(table_name)
table.schema = (
        bq.SchemaField('Label',            'STRING'),
        bq.SchemaField('Description',      'STRING'),
)
client.create_table(table)
```

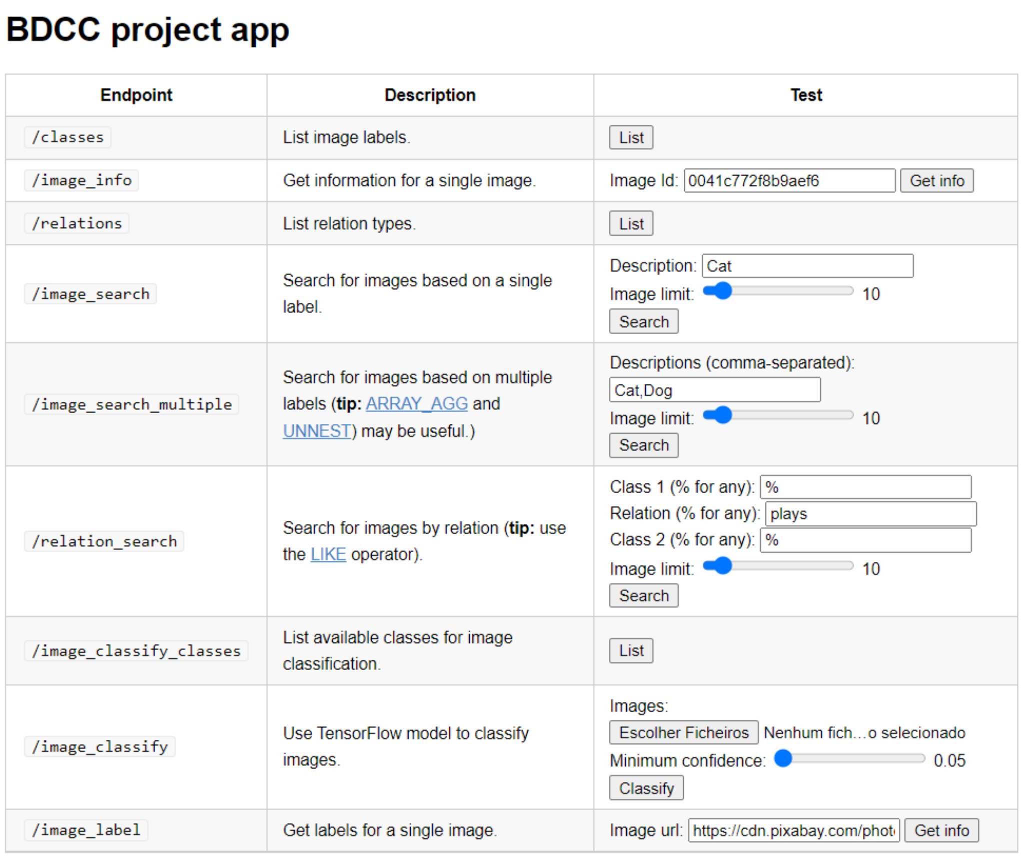
Followed by the upload of the csv to the BigQuery table:

```
client.load_table_from_dataframe(classes, table)
```

App Engine

App Engine is a platform as service (Paas) for developing and hosting web applications at scale https://cloud.google.com/appengine/docs. The program endpoints are specified in *main.py* and the *app.yaml* specifies the runtime environment. Also, the html files for the corresponding endpoints are in the *templates* folder. The overall structure and layout of the app is defined in *index.html*.

Application layout



BDCC project app

Endpoint	Description	Test
`/classes`	List image labels.	List
`/image_info`	Get information for a single image.	Image Id: 0041c772f8b9aef6 Get info
`/relations`	List relation types.	List
`/image_search`	Search for images based on a single label.	Description: Cat Image limit: 10 Search
`/image_search_multiple`	Search for images based on multiple labels (**tip:** ARRAY_AGG and UNNEST) may be useful.)	Descriptions (comma-separated): Cat,Dog Image limit: 10 Search
`/relation_search`	Search for images by relation (**tip:** use the LIKE operator).	Class 1 (% for any): % Relation (% for any): plays Class 2 (% for any): % Image limit: 10 Search
`/image_classify_classes`	List available classes for image classification.	List
`/image_classify`	Use TensorFlow model to classify images.	Images: Escolher Ficheiros Nenhum fich...o selecionado Minimum confidence: 0.05 Classify
`/image_label`	Get labels for a single image.	Image url: https://cdn.pixabay.com/phot Get info

Newly implemented

The basic functioning of the implemented Flask application consists in storing the received user inputs using the *flask.request.args.get* function as objects, and then performing queries using these elements.

- **image_search_multiple**

  Displays the image ID and the image of the selected number of pictures based on the description by class. The BigQuery query does this by converting the input array to a table with each element of the array corresponding to a row.

```
results = BQ_CLIENT.query(
'''
    SELECT ImageId
    FROM `project-bdcc-nuno.openimages.image_labels`
    JOIN `project-bdcc-nuno.openimages.classes` USING(Label)
    WHERE Description IN UNNEST({0})
    ORDER BY ImageId
    LIMIT {1}
'''.format(descriptions, image_limit)
).result()
```

- **relations**

  Provides the available relations and the corresponding absolute frequency. The query does this using the GROUP BY SQL function.

```
results = BQ_CLIENT.query(
'''
    Select Relation, COUNT(*) AS NumImages
    FROM `project-bdcc-nuno.openimages.relations`
    GROUP BY Relation
    ORDER BY Relation
''').result()
```

  However, I was unable to configure the hyperlink for showing all the available images for that relation.

- **relations_search**

  This endpoint should receive 3 inputs: class1 (label); relation; and class2 (label) and display all the images with this kind of relationship. However, I was unable to perform this query probably due to not paying close attention to the keys in the data model.

- **image_info**

  Receives an image ID and shows the image, the classes and the relations. To perform this query, I joined the three datasets and searched for imageId:

```
results = BQ_CLIENT.query(
'''
    SELECT DISTINCT Description, ImageId
    FROM `project-bdcc-nuno.openimages.image_labels`
    JOIN `project-bdcc-nuno.openimages.classes` USING(Label)
    JOIN `project-bdcc-nuno.openimages.relations` USING(ImageId)
    WHERE ImageId = '{0}'
    ORDER BY Description
'''.format(image_id)
).result()
```

Despite having achieved to show the classes present in the image, I failed to present the correct relations, probably should have paid closer attention to the GroupBy function.

- **image_classify**

The image classification endpoint classifies an uploaded image using an AI algorithm. The ten images' classes available for classification could be seen in the *image_classify_classes* endpoint.

The AI model was built using transfer learning of a previously deep learning model trained by Google. The Keras model learnt the adequate weights for our specific problem using 79 images for the training sample, 20 for the validation set and 11 for the test set. Images were uniformly distributed by classes. Should have been a 8:1:1 ration, but I made a logic mistake when creating the dataset.

The python script for the *automl.csv* creation is available as *AutoML.ipynb*. Mainly, I queried over the classes of interest and saved the results as a dataframe object.

```
# Save output in a variable `df`

%%bigquery --project project-bdcc-nuno df
SELECT *
FROM `project-bdcc-nuno.openimages.image_labels`
JOIN `project-bdcc-nuno.openimages.classes` USING(Label)
WHERE Description IN ('Bowl', 'Pig', 'Tank', 'Leopard', 'Umbrella',
'Fox', 'Lobster', 'Whale', 'Skateboard', 'Hamster')
```

Then, I converted the dataframe to a list and iterated over the ten possible classes by storing the class name and the corresponding sample set ("train", "validation", "test").

```
new = []
contador2 = 0
for val in ['Bowl', 'Pig', 'Tank', 'Leopard', 'Umbrella', 'Fox',
'Lobster', 'Whale', 'Skateboard', 'Hamster']:
  contador = 0
  for row in range(len(df2)):
    #select 100 images of each class
    if df2[row][2] == val and contador <100:
      new.append(df2[row])
      contador+=1

      #append TRAIN-VALIDATION-TEST SPLIT
      if contador < 80:
        new[contador2].append("TRAIN")
      elif contador < 90:
        new[contador2].append("VALIDATION")

      else:
        new[contador2].append("TEST")

      contador2+=1
```

Then, changed the column order and pointed to the image bucket:

```
df3 = df3.iloc[:, [3,1,2]]
# point to image bucket
df3.iloc[:,1] = df3.iloc[:,1].apply(lambda x:
"gs://open_images_nuno/images_1000/" + x + ".jpg")
```

Finally, exported the results to a csv file.

To save the selected image for uploading to the bucket consisted of the images and csv file:

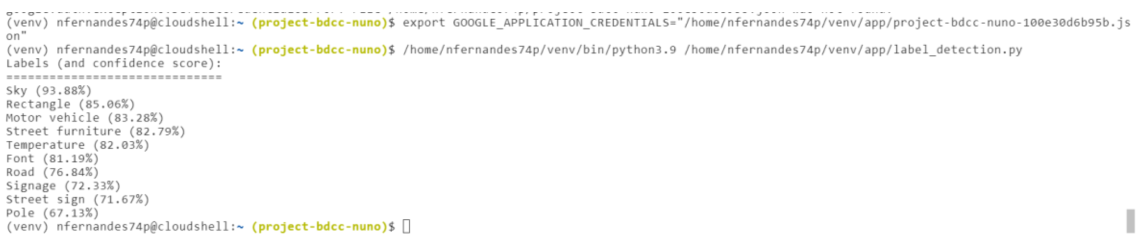
```
from google.cloud import storage

"""Downloads a public blob from the bucket."""
# bucket_name = "your-bucket-name"
# source_blob_name = "storage-object-name"
# destination_file_name = "local/path/to/file"
storage_client = storage.Client.create_anonymous_client()

bucket = storage_client.bucket("bdcc_open_images_dataset")

#save only selected images
for row in range(len(new)):
  blob = bucket.blob("images/" + new[row][1] + ".jpg")
  blob.download_to_filename("images_1000/" + new[row][1] + ".jpg")
```

After training the model for the offline use, one adds to the app's static folder the *dict.txt* file with the 10 available classes for classification and the *model.tflite* file with the model's weights.

- **image_label**

The last endpoint should display the labels/category types of an image using Vision pre-trained machine learning models. Despite I was able to use the API as an end-user by creating a json key, I did not create the html template successfully, and I could not deploy the application (I suppose it is due to a bad permission configuration, when creating the key).

In case the permission was not denied and the template was well configured, the user would inserted the image URL and a page in the web browser would show the labels present in the image (*image_label.html*).



```
(venv) nfernandes74p@cloudshell:~ (project-bdcc-nuno)$ export GOOGLE_APPLICATION_CREDENTIALS="/home/nfernandes74p/venv/app/project-bdcc-nuno-100e30d6b95b.js
on"
(venv) nfernandes74p@cloudshell:~ (project-bdcc-nuno)$ /home/nfernandes74p/venv/bin/python3.9 /home/nfernandes74p/venv/app/label_detection.py
Labels (and confidence score):
==============================
Sky (93.88%)
Rectangle (85.06%)
Motor vehicle (83.28%)
Street furniture (82.79%)
Temperature (82.03%)
Font (81.19%)
Road (76.84%)
Signage (72.33%)
Street sign (71.67%)
Pole (67.13%)
(venv) nfernandes74p@cloudshell:~ (project-bdcc-nuno)$
```

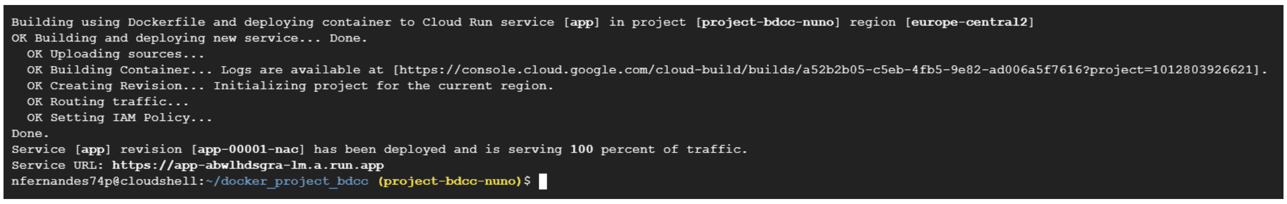
Compute Engine

Contrarily to AppEngine, Compute Engine is an Infrastructure as a service (IaaS). Therefore, one must explicitly define our container, which was achieved by creating a Docker image. To do that, I copied the app folder and renamed it to *docker_project_bdcc. Then,* I deleted the *app.yaml* file and added *gunicorn* and *google-cloud-datastore* to the *requirements.txt* file. Finally, I created a *Dockerfile*:

```
#base containter
FROM python:3-slim
#directory
WORKDIR /docker_project_bdcc
#copy all the source code
COPY . .
#install the requirements
RUN pip install -r requirements.txt
#start gunicorn to serve the app
ENTRYPOINT exec gunicorn -b :$PORT -w 2 main:app
```

Then, deployed the app: $ gcloud run deploy app - - source .

Despite receiving the message that the service was running, I got an error when I accessed the app. https://app-abwlhdsgra-lm.a.run.app/



```
Building using Dockerfile and deploying container to Cloud Run service [app] in project [project-bdcc-nuno] region [europe-central2]
OK Building and deploying new service... Done.
  OK Uploading sources...
  OK Building Container... Logs are available at [https://console.cloud.google.com/cloud-build/builds/a52b2b05-c5eb-4fb5-9e82-ad006a5f7616?project=1012803926621].
  OK Creating Revision... Initializing project for the current region.
  OK Routing traffic...
  OK Setting IAM Policy...
Done.
Service [app] revision [app-00001-nac] has been deployed and is serving 100 percent of traffic.
Service URL: https://app-abwlhdsgra-lm.a.run.app
nfernandes74p@cloudshell:~/docker_project_bdcc (project-bdcc-nuno)$
```