PaLM API (Embedding) and Vertex Matching Engine

By Shivaji Dutta May 10, 2023

In this document we will step through a practical example of building a fast matching engine service on Vertex AI, building Embedding using the new Vertex AI PaLM API.

We will use the IMDB reviews dataset as an example as it has good amounts of publicly available texts.

The document will cover the following

- Load the IMDB Dataset
- Use Google LLM Embedding API to generate Embedding
- Save the Embedding into a File
- Load the Embeddings SCANN API
- Query the SCANN API
- Setup Vertex AI Matching Engine Public Endpoint
- Create an Public Index
- Deploy the Index to an Public Endpoint
- Query the Public Index Endpoint

Objective

The objective of this tutorial is to learn a combination of PaLM API and Matching Engine Service (SCANN and Vertex), to use them for other use cases, rather than explore the dataset of IMDB Movie Reviews.

Dataset

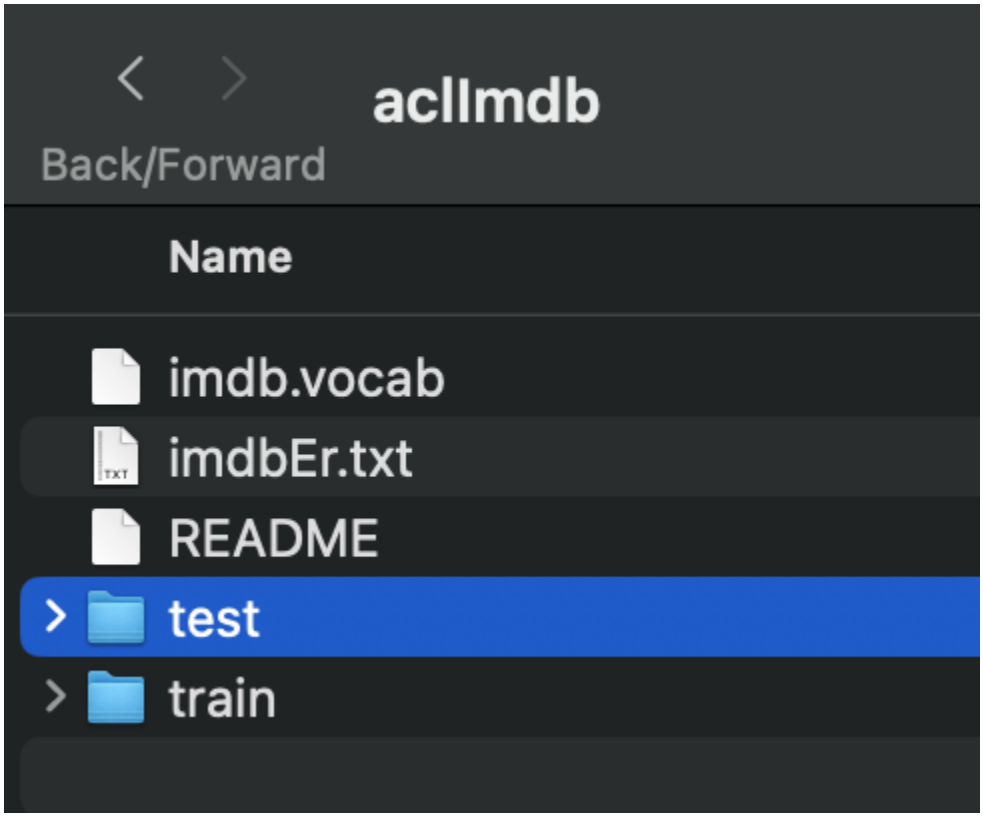
IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly popular movie reviews

for training and 25,000 for testing. So, predict the number of positive and negative reviews using either classification or deep learning algorithms.

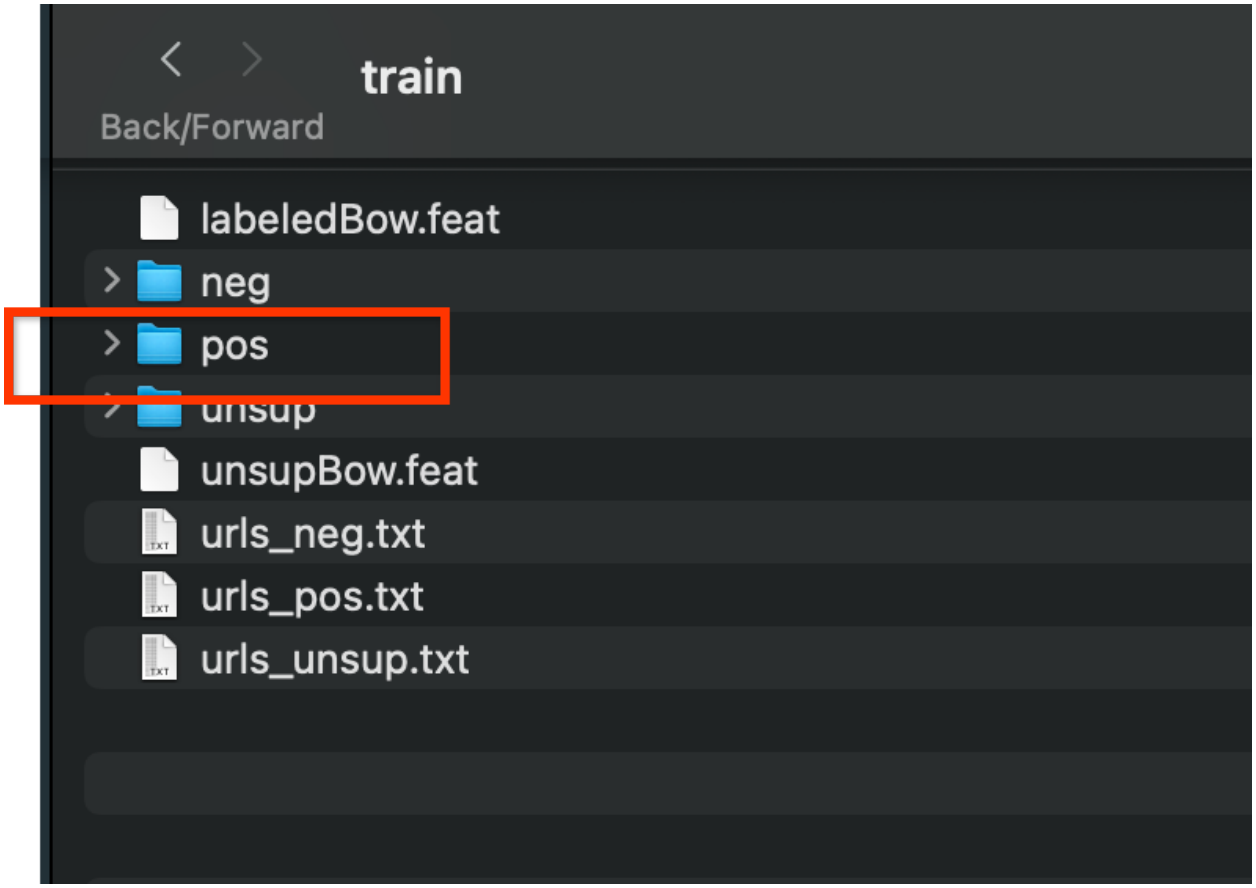
For more dataset information, please go through the following link,

http://ai.stanford.edu/~amaas/data/sentiment/

Please download and store the dataset locally. We will use it later on to extract the positive reviews. The downloaded file called "aclImdb_v1.tar.gz" is about 84.1 MB.
Once you unzip the dataset, you should see the following structure.



This is a typical classification dataset. We will use a subset of the dataset, POSITIVE reviews from the **train** folder for our Matching Engine. This has about 12,500 review text files. You could use all the dataset, it does not matter, just for the tutorial I presumed that 12,500 was a good enough size.



We will use the above data later. Before we get into the details about the dataset, let us explore the ner Vertex PALM API for Embedding Generation

Embedding Generation using Vertex PaLM API

Vertex PALM API is in private preview. Let us start with installing the API. It is suggested that you could use a Google Colab notebook for this.

Authenticate

```python
from google.colab import auth as google_auth
google_auth.authenticate_user()
```

Copy the private binary over:

```python
!pip install google-cloud-aiplatform --upgrade --user
```

- Restart runtime after install

Now you are ready to use the API.

```python
import vertexai
from vertexai.preview.language_models import
TextEmbeddingModel


model =
TextEmbeddingModel.from_pretrained("textembedding-gecko@001")
```

Now you are ready to use the model.

We will start with two hello world texts, and test the API to get an embedding vector

a) Hello, how are you?
b) What is your name?

You could use any other texts. The idea is to just understand the embedding API and what it is.

The embedding API takes a python List object. This is how we will invoke the API

```
Python
embeddings = model.get_embeddings(["Hello how are you?","What is your name?"])
```

The API returns a List of Embedding vectors. Each vector is of size 768. It takes a token size of 3,072 input tokens and outputs 768-dimensional vector embeddings.

```
Python
emb1 = embeddings[0]
emb2 = embeddings[1]
```

Each of the elements of the embedding returns an object of the following class.

```
Unset
vertexai.language_models._language_models.TextEmbedding
```

To get access to the vector values, access the “values” property

Python

```
emb1.values
```

IMDB Dataset Embedding Generation

Now that we have learnt about the Embedding API. Now lets use the above API to generate Embeddings for all the texts in the IMDB dataset. This should be fairly straight forward.

Since we would want to use the dataset finally in the Vertex Matching Engine Dataset, I think we should understand a bit about the format of the data it is expecting.

This document talks about the structure of the data. You can prepare the data in CSV, JSON or AVRO format. The product document discusses details for all the three formats.

For this tutorial we will follow the CSV format. The CSV format expects the data in the following format.

The first column is an unique id and the subsequent columns are embedding values of the text. In our case since we will be using the TextEmbedding API which returns 768 vector, we will have 768 columns of the data.

Unset

```
Id, val1, val2,,,val768
```

Example of 1 row of data in the screenshot below

Let us Upload the data into your colab notebook from your local machine that you downloaded earlier.

```python
from google.colab import files
uploaded = files.upload()
```

Unzip the files

```python
!tar -xzvf 'aclImdb_v1.tar.gz'
```

List the positive reviews folder

```python
!ls aclImdb/train/pos
```

Extract the positive reviews in list “rev”

```python
import os
rev = []

for file in files:
  with open('aclImdb/train/pos/'+file, 'r') as f:
      str = f.read()
      rev.append(str)
```

Next, let us generate embeddings of the reviews. The API currently supports max of 5 texts per invocation. So we batch it to 5.

This will take some time to generate 12,500 embeddings

```python
index = 0
allembeddings = []
for i in range(5,len (rev)+1,5):
   print(i)
   embedarr = model.get_embeddings(rev[index:i] )
   allembeddings.append(embedarr)
   index = i
```

Since the above code generates a list of arrays, we need to convert it into a flat list

```python
flatlist = []
for l in allembeddings:
   for e in l:
       flatlist.append(e.values)
```

Load the text (reviews) and embeddings into a dataframe.

```python
import pandas as pd
df = pd.DataFrame()
df["embeddings"] = flatlist
df["reviews"] = rev
```

Lets save the dataframe, so that we can use the embeddings later on

```python
df.to_csv("imdb_embeddings.csv")
```

Download a local copy for accessing later on:

```python
from google.colab import files
files.download("imdb_embeddings.csv")
```

We are going to have another version of the dataframe where we will only have the id, and the embeddings columns. This will form as an input to the Vertex Matching Engine document.

```python
medf = pd.read_csv("imdb_embeddings.csv")
```

```
medf.drop("Unnamed: 0", axis=1, inplace=True)
medf.drop("reviews",axis=1, inplace=True)
medf.head()
```

Save it and download it for use in later sections.

Python
```
medf.to_csv("matching_engine_imdb_embeddings.csv")
```

Download a local copy for accessing later on:

Python
```
from google.colab import files
files.download("matching_engine_imdb_embeddings.csv")
```

Matching Engine Using SCANN API

ScaNN (Scalable Nearest Neighbors) is a method for efficient vector similarity search at scale. ScaNN achieves state-of-the-art performance on ann-benchmarks.com.

In the following section we will use the Movie Reviews Embedding generated above to use with SCANN Matching engine to look up closest movie reviews in the 12,500 in the dataset.

Install SCANN API

We will start by installing the Python API

```python
!pip install scann
```

Let's use the SCANN API

```python
import scann
import numpy as np

#Normalize the flatlist dataset

normalized_dataset = flatlist / np.linalg.norm(flatlist,
axis=1)[:, np.newaxis]

#Load the embedding list (flatlist) into the SCANN Index,

searcher = scann.scann_ops_pybind.builder(normalized_dataset,
10, "dot_product").tree(
   num_leaves=2000, num_leaves_to_search=100,
training_sample_size=250000).score_ah(
   2,
anisotropic_quantization_threshold=0.2).reorder(100).build()
```

Create a utility Search Function to query the index. Following function takes

```python
import time
def search_posts(query):
   start = time.time()
   query = model.get_embeddings([query])[0].values
   neighbors, distances = searcher.search(query,
final_num_neighbors=3)
   end = time.time()
   return neighbors, distances, end-start
```

Let us test some review matches. We are going to use some texts from the training dataset for now. You can use any text as you please.

I am randomly selecting the 10th review.

```python
query = model.get_embeddings([rev[10]])[0].values
neighbors, distances = searcher.search(query,
final_num_neighbors=3)
```

You should see the following response:

```
(array([  10, 4246, 3672], dtype=uint32),
 array([0.9999957, 0.8866681, 0.8785361], dtype=float32))
```

The first array is of the 3 indexes it returns and the next array is the confidence scores of the matches

You can inspect the original text and the neighbors in the following manner:

```python
print ("Original post:", rev[10])
print ("Neighbor 1:", rev[4246])
```

Or you could just use a small snippet of code to list the text of all the neighbors:

```python
for i in range (len(neighbors)):
 print("**")
 print(i, neighbors[i])
 print(rev[neighbors[i]])
 print("**")
```

Now that we have tested the embedding generation with SCANN nearest neighbor.

Let's move on to use the Vertex Matching Engine

Vertex Matching Engine Public Endpoints

In the next few steps we will use the Vertex Matching Engine Public Endpoints. This is a new feature and makes it very easy to test the Matching Engine without worrying about Network Peering, VPC etc.

In the previous steps I have created a CSV file which has all the 12,500 embeddings of the reviews. I saved it here for ease of use. Please download the copy of it.

We will change the format so that it can be consumed by the Vertex Matching Engine Index.

```python
import pandas as pd
df = pd.read_csv("matching_engine_imdb_embeddings.csv")
testdf = df
with open("matching_engine_imdb_embeddings_updated.csv", "w")
as f:
    for i in range(testdf.embedding.size):
        f.write( str(i) + ',')
        f.write(
",".join(testdf.embedding[i][1:-1].split(",")))
        f.write("\n")
```

The file should look like this:

You can test a sample embedding:

```python
df.embedding[10]
```

Now that the file is ready, we need to place it in a Google Cloud Bucket so that we can start to index it. Copy the file (`matching_engine_imdb_embeddings_updated.csv`) over to the following `BUCKET_ROOT` location.

Set the BUCKET_ROOT env variable.

```python
BUCKET_ROOT = "gs://vertex-matching-engine-sd/imdb_root/"
```

Index Creation

We will also set a bunch of environment variables that will be needed during the index generation.

```python
PROJECT_ID = "demogct2022"
REGION = "us-central1"
ENDPOINT = "{}-aiplatform.googleapis.com".format(REGION)
NETWORK_NAME = "ucaip-haystack-vpc-network"  # @param {type:"string"}


AUTH_TOKEN = !gcloud auth print-access-token
PROJECT_NUMBER = !gcloud projects list --filter="PROJECT_ID:'{PROJECT_ID}'" --format='value(PROJECT_NUMBER)'
PROJECT_NUMBER = PROJECT_NUMBER[0]

PARENT = "projects/{}/locations/{}".format(PROJECT_ID, REGION)

print("ENDPOINT: {}".format(ENDPOINT))
print("PROJECT_ID: {}".format(PROJECT_ID))
print("REGION: {}".format(REGION))

!gcloud config set project {PROJECT_ID}
!gcloud config set ai_platform/region {REGION}
```

Python

```
from google.cloud import aiplatform_v1beta1
from google.protobuf import struct_pb2
import time

index_client = aiplatform_v1beta1.IndexServiceClient(
    client_options=dict(api_endpoint=ENDPOINT)
)

DIMENSIONS = 768
DISPLAY_NAME = "imdb_reviews_768_1"
```

Set Index configuration for the index

Python

```
treeAhConfig = struct_pb2.Struct(
    fields={
        "leafNodeEmbeddingCount":
struct_pb2.Value(number_value=500),
        "leafNodesToSearchPercent":
struct_pb2.Value(number_value=7),
    }
)


algorithmConfig = struct_pb2.Struct(
    fields={"treeAhConfig":
struct_pb2.Value(struct_value=treeAhConfig)}
)


config = struct_pb2.Struct(
```

```
    fields={
        "dimensions":
struct_pb2.Value(number_value=DIMENSIONS),
        "approximateNeighborsCount":
struct_pb2.Value(number_value=150),
        "distanceMeasureType":
struct_pb2.Value(string_value="DOT_PRODUCT_DISTANCE"),
        "algorithmConfig":
struct_pb2.Value(struct_value=algorithmConfig),
    }
)


metadata = struct_pb2.Struct(
    fields={
        "config": struct_pb2.Value(struct_value=config),
        "contentsDeltaUri":
struct_pb2.Value(string_value=BUCKET_ROOT),
    }
)

ann_index = {
    "display_name": DISPLAY_NAME,
    "description": "IMDB POSITIVE REVIEWS ANN index",
    "metadata": struct_pb2.Value(struct_value=metadata),
}

ann_index = index_client.create_index(parent=PARENT,
index=ann_index)


#Poll to check the index creation. This can take upto 45 mins.
while True:
    if ann_index.done():
```

```
    break
print("Poll the operation to create index...")
time.sleep(60)
```

Once Done you can check Index Created.

```
Python
INDEX_RESOURCE_NAME = ann_index.result().name
INDEX_RESOURCE_NAME
```

You can navigate to the Matching Engine UI and view your index created. Note the value of the index id, it will be needed later to deploy the index.

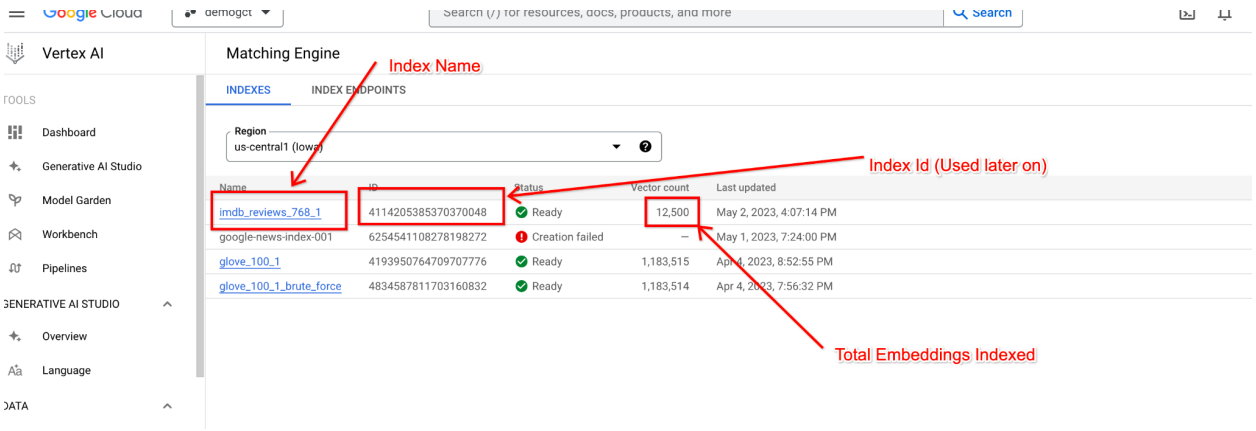


Fig: Above is a screenshot of the Vertex Matching Engine Node.

Deploy the Index to a Public Endpoint

Deploying an index includes the following three tasks:

1. Create an `IndexEndpoint`
2. Get the `IndexEndpoint` ID.
3. Deploy the index to the `IndexEndpoint`.

In this section we will use CURL commands and request json (handcrafted) to Deploy the index.

Create an `IndexEndpoint`

To use a public endpoint, set the `publicEndpointEnabled` field to true.

Lets create a **request.json** as the following. We name the endpoint as imbd-public-endpoint.

```
Unset

{
  "display_name": "imdb-public-endpoint",
"publicEndpointEnabled":"true"

}
```

Next we need to form the URL to the post. Which takes the following format

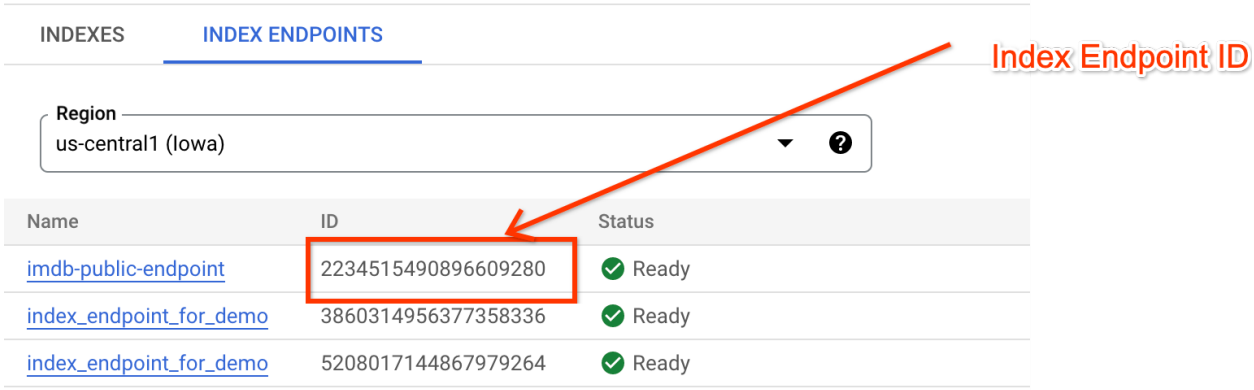
```
curl -X POST \
    -H "Authorization: Bearer $(gcloud auth print-access-token)" \
    -H "Content-Type: application/json; charset=utf-8" \
    -d @request.json \
    "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION/
```

Replace the location and the project id:
Location = us-central1
Project = <your projectid>

In my case:

```
Unset
curl -X POST     -H "Authorization: Bearer $(gcloud auth
print-access-token)"     -H "Content-Type: application/json;
charset=utf-8"     -d @request.json
"https://us-central1-aiplatform.googleapis.com/v1/projects/dem
ogct2022/locations/us-central1/indexEndpoints"
```

This should start the index endpoint creation. You can check its status in the UI. Once it is ready it should have an ID assigned to it.



Next we will Deploy the IMDB Review Index to the above Endpoint.

Before using any of the request data, prepare the following values.

- **LOCATION**: The region where you are using Vertex AI.
- **PROJECT**: Your project ID.
- **INDEX_ENDPOINT_ID**: The ID of the index endpoint.

- **DEPLOYED_INDEX_ID**: A user specified string to uniquely identify the deployed index. It must start with a letter and contain only **letters, numbers or underscores**. See DeployedIndex.id for format guidelines.
- **DEPLOYED_INDEX_NAME**: Display name of the deployed index.
- **INDEX_ID**: The ID of the index.
- **PROJECT_NUMBER**: Project number for your project

We will create a new request-deploy.json file based on the template below.

Sample file, that I have used in my project.

```
Python
{
  "deployedIndex": {
    "id": "imdb_ann_public_endpoint",
    "index":
"projects/demogct2022/locations/us-central1/indexes/4114205385
370370048",
    "displayName": "public endpoint for imdb index service"
  }
}
```

Next Post the above request-deploy.json file to the following location using CURL

Save the request body in a file called `request.json`, and execute the following command:

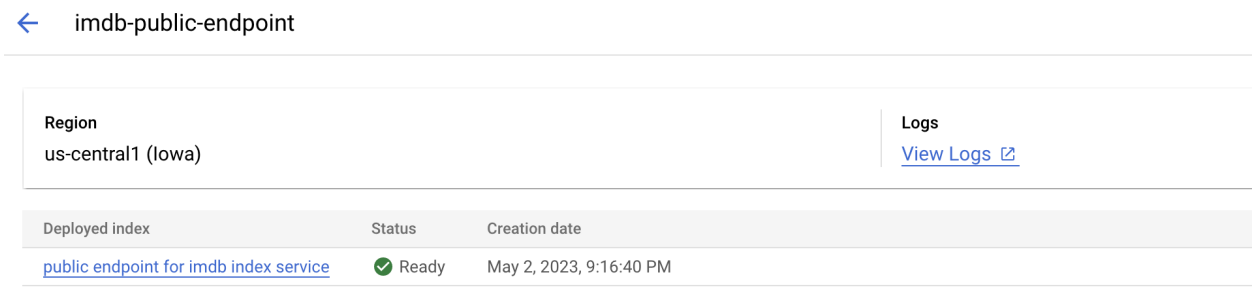
```
curl -X POST \
    -H "Authorization: Bearer $(gcloud auth print-access-token)" \
    -H "Content-Type: application/json; charset=utf-8" \
    -d @request.json \
    "https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT/locations/LOCATION
```

Sample

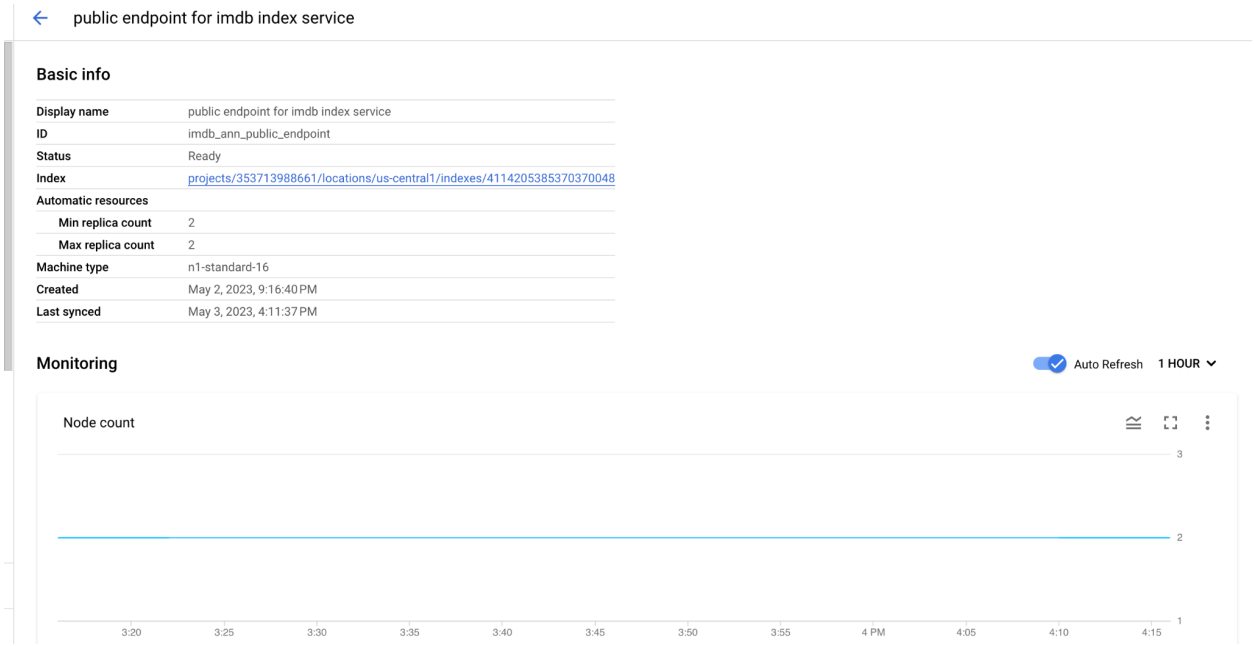
Unset

```
curl -X POST     -H "Authorization: Bearer $(gcloud auth
print-access-token)"     -H "Content-Type: application/json;
charset=utf-8"     -d @request-deploy.json
"https://us-central1-aiplatform.googleapis.com/v1/projects/dem
ogct2022/locations/us-central1/indexEndpoints/2234515490896609
280:deployIndex"
```

You can check the GCP Vertex Matching Engine UI to see if the index is deployed. This can take anywhere from 15-30 mins.



You can click the deployed index to view more details



If you are curious, you can also see a new VM deployed in the GCP VM console.

Now that your index is deployed you are ready to run your queries. This is a one time task that you don't need to do again and again. For production, perhaps you can automate this process.

To access the endpoint you need a publicendpointDomainName

Get the index domain name

```
Unset
curl -H "Content-Type: application/json" -H "Authorization: Bearer
`gcloud auth print-access-token`"
${ENDPOINT}/v1/projects/${PROJECT_ID}/locations/${REGION}/indexEndpo
ints/${INDEX_ENDPOINT_ID}
```

Sample used for my project (demogct2022)

Unset

```
curl -H "Content-Type: application/json" -H "Authorization:
Bearer `gcloud auth print-access-token`"
"https://us-central1-aiplatform.googleapis.com/v1/projects/dem
ogct2022/locations/us-central1/indexEndpoints/2234515490896609
280"
```

This should return

Python

```
{
  "name":
"projects/353713988661/locations/us-central1/indexEndpoints/22
34515490896609280",
  "displayName": "imdb-public-endpoint",
  "deployedIndexes": [
    {
      "id": "imdb_ann_public_endpoint",
      "index":
"projects/353713988661/locations/us-central1/indexes/411420538
5370370048",
      "displayName": "public endpoint for imdb index service",
      "createTime": "2023-05-03T04:16:40.799366Z",
      "indexSyncTime": "2023-05-03T23:16:37.273465Z",
      "automaticResources": {
        "minReplicaCount": 2,
        "maxReplicaCount": 2
      },
      "deploymentGroup": "default"
    }
  ],
```

```
  "etag":
"AMEw9yMmOxlxab0Ag26oE3h0zmlY-0SNFXtEU0WAVV8UiwVYX4xBlVDFhgPZ7
AH6xF__",
  "createTime": "2023-05-03T04:03:20.275594Z",
  "updateTime": "2023-05-03T04:03:21.641783Z",
  "publicEndpointDomainName":
"1576917689.us-central1-353713988661.vdb.vertexai.goog"
}
```

We will grab the **publicEndpointDomainName** to be used by our Query Code. In the above it is **"1576917689.us-central1-353713988661.vdb.vertexai.goog".**

Security

The Public endpoints are secured. To access we need a Service Account Token with "Vertex AI User" role. Create a Service account and download the key. We will use the key to access the Matching Engine Service.

Query

In the following code snippet which helps in querying the endpoint using python. The template code is from the documentation. Change the following:

- **Sa_file_path - Path of the security key**
- **Client_options - publicEndpointDomainName**
- **Index_endpoint - Your index endpoint created above**
- **Deployed_index_id - Name of the deployed Index Id.**

Python

```
from google.oauth2 import service_account
```

```
from google.cloud import aiplatform_v1beta1

def findneighbor_sample( val):
  # The AI Platform services require regional API endpoints.
  scopes = ["https://www.googleapis.com/auth/cloud-platform"]

  # create a service account with `Vertex AI User` role
granted in IAM page.
  # download the service account key
https://developers.google.com/identity/protocols/oauth2/servic
e-account#authorizingrequests
  sa_file_path = "/content/demogct2022-ca8b4e443d11.json"

  credentials =
service_account.Credentials.from_service_account_file(
      sa_file_path, scopes=scopes
  )
  client_options = {
      "api_endpoint":
"1576917689.us-central1-353713988661.vdb.vertexai.goog"
  }

  vertex_ai_client = aiplatform_v1beta1.MatchServiceClient(
      credentials=credentials,
      client_options=client_options,
  )

  request = aiplatform_v1beta1.FindNeighborsRequest(

index_endpoint="projects/353713988661/locations/us-central1/in
dexEndpoints/2234515490896609280",
      deployed_index_id="imdb_ann_public_endpoint",
  )
  dp1 = aiplatform_v1beta1.IndexDatapoint(
```

```
        datapoint_id="0",
        feature_vector=val,
    )
    query = aiplatform_v1beta1.FindNeighborsRequest.Query(
        datapoint=dp1,
    )
    request.queries.append(query)

    response = vertex_ai_client.find_neighbors(request)
    print("Now")
    print(response)
```

Test the above function

We take a sample text, convert it to an embedding using the API and use it to find neighbors

Python

```
text = "My boyfriend and I went to watch The Guardian.At first
I didn't want to watch it, but I loved the movie- It was
definitely the best movie I have seen in sometime.They
portrayed the USCG very well, it really showed me what they do
and I think they should really be appreciated more.Not only
did it teach but it was a really good movie. The movie shows
what the really do and how hard the job is.I think being a
USCG would be challenging and very scary. It was a great movie
all around. I would suggest this movie for anyone to see.The
ending broke my heart but I know why he did it. The storyline
```

```
was great I give it 2 thumbs up. I cried it was very
emotional, I would give it a 20 if I could!"

val = model.get_embeddings([text])[0].values
findneighbor_sample(val)
```

Congratulations!! You just finished using the Matching Engine with PaLM TextEmbedding API to find the matching reviews to your reviews.