To create eks clusters and deploy LLM and embedding encoders:

```
eksctl create cluster \
  --name rag-cluster1 \
  --version 1.28 \
  --region us-east-1 \
  --nodegroup-name rag-nodes1 \
  --node-type m5.xlarge \
  --nodes 1 \
  --nodes-min 1 \
  --nodes-max 2 \
  --managed
```

```
aws ecr create-repository --repository-name rag-generator
```

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
540009924757.dkr.ecr.us-east-1.amazonaws.com
```

```
docker tag rag_generator0.4.3:latest
540009924757.dkr.ecr.us-east-1.amazonaws.com/rag-generator:latest
```

```
docker push 540009924757.dkr.ecr.us-east-1.amazonaws.com/rag-generator:latest
```

Create AMD64 processor image from a Mac through buildx

```
docker buildx create --name mybuilder --use
docker buildx inspect --bootstrap
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
540009924757.dkr.ecr.us-east-1.amazonaws.com
docker buildx build --platform linux/amd64,linux/arm64 -t
540009924757.dkr.ecr.us-east-1.amazonaws.com/rag-generator:0.1 . --push
```

Get endpoints after deploying

```
kubectl apply -f flask-app-service.yaml
kubectl get svc flask-app-service
```

Embedding extractor test

```
import requests
dataset = []
# The URL where your Flask app is running
# Change the port if necessary, depending on how you mapped the ports in the Docker
container
url = ''
```

```
# Example data to send
data = {
   'chunks': dataset
}

# Send a POST request
response = requests.post(url, json=data)

# Check if the request was successful
if response.status_code == 200:
   print('Response from server:', response.json())
else:
   print('Error:', response.status_code)
```

LLM Inference

```
url = ''

# Example data to send
data = {
   'question': question,
   "context": context
}

# Send a POST request

headers = {
        "Content-Type": "application/json",
        "Api-Key": os.environ.get('api_key')
    }
print("Pre passing to model")
print(data)
response = requests.post(llm_host, json=data, headers=headers)


# Check if the request was successful
if response.status_code == 200:
   print('Response from server:', response.json())
else:
   print('Error:', response.status_code)
```

Using LLM, Embedding Extractor, Opensearch for RAG

```
import requests
import os



question = "what is the entry point for requests?"
query_vector, question = get_single_embedding(question)

k = 1  # Number of nearest neighbors to find

knn_query = {
   "size": 1,  # Adjust 'size' as needed
   "query": {
       "bool": {
           "must": {
               "knn": {
                   "embedding": {
                       "vector": query_vector,  # Replace with your vector values
                       "k": 1  # Adjust 'k' as needed
                   }
               }
           },
           "filter": {
               "term": {"doc_id": ""}  # Replace with your specific doc_id
           }
       }
   }
}

response = client.search(index=index_name, body=knn_query)



documents = response['hits']['hits']
context = ''
# Optionally, process the results
for doc in documents:
   print(doc['_score'])
   print(doc['_source']['text'])  # Replace with your processing logic
   print('----')
   context += doc['_source']['text']
```

```
# The URL where your Flask app is running
# Change the port if necessary, depending on how you mapped the ports in the Docker
container
url = ''

# Example data to send
data = {
   'question': question,
   "context": context
}
# Send a POST request
headers = {
   "Content-Type": "application/json",
   "Api-Key": os.environ.get(‘api_key’)
}
print("Pre passing to model")
print(data)
response = requests.post(url, json=data, headers=headers)

# Check if the request was successful
if response.status_code == 200:
   print('Response from server:', response.json())
else:
   print('Error:', response.status_code)
```