**Image Recognition System: Installation and Operation Guide**

This document provides a step-by-step guide for setting up and operating the Image Recognition System on your own laptop. Our system enables users to upload images for analysis, where a custom image recognition model deployed on the IBM Cloud platform processes these images and provides recognition results. This guide will walk you through the installation process and explain how the system works locally.

**Table of Contents**

1. System Requirements
2. Installation Steps
3. System Operation
4. Accessing the User Interface
5. Image Upload and Recognition
6. Providing Feedback
7. System Cleanup

**1. System Requirements**

Before you begin the installation process, ensure that your laptop meets the following requirements:

* Operating System: Windows, macOS, or Linux

**Dockerfile:**

# Use the official Python image as the base image

FROM python:3.8

# Set the working directory in the container

WORKDIR */app*

# Copy the requirements file into the container at /app

COPY *requirements.txt* */app/*

# Install any needed packages specified in requirements.txt

RUN *pip* *install* *-r* *requirements.txt*

# Copy your Flask application code into the container

COPY *.* */app/*

# Expose port 5000 for the Flask app

EXPOSE *5000*

# Define the command to run your Flask application

CMD *[*"python"*,* "app.py"*]*

**requirements.txt :**

Flask==2.1.1

tensorflow==2.4.1

tensorflow-hub==0.12.0

matplotlib==3.4.3

Pillow==8.0.0

ibm-cos-sdk==2.10.0

**2. Installation Steps**

Follow these steps to install and set up the Image Recognition System on your laptop:

**Step 1: Clone the Repository**

* Open a terminal or command prompt.
* Navigate to the directory where you want to install the system.
* Run the following command to clone the system repository from GitHub:

bashCopy code

git clone https://github.com/yourrepositoryurl.git

**Step 2: Create a Virtual Environment (Optional, but Recommended)**

* Navigate to the cloned repository's directory.
* Create a virtual environment using the following command:

bashCopy code

python -m venv venv

* Activate the virtual environment:
  + **Windows:**

bashCopy code

venv\Scripts\activate

* + **macOS and Linux:**

bashCopy code

source venv/bin/activate

**Step 3: Install Required Packages**

* Install the necessary Python packages using **pip**:

bashCopy code

pip install -r requirements.txt

**Step 4: Run the System**

* Start the system by running the following command:

bashCopy code

python app.py

The system should now be running locally on your laptop.

**3. System Operation**

The Image Recognition System operates as follows:

* Users interact with the system through a user-friendly web interface.
* Users can upload images for analysis.
* Uploaded images are processed by the locally deployed image recognition model.
* Recognition results, including recognized content and accuracy levels, are displayed to the user.

**4. Accessing the User Interface**

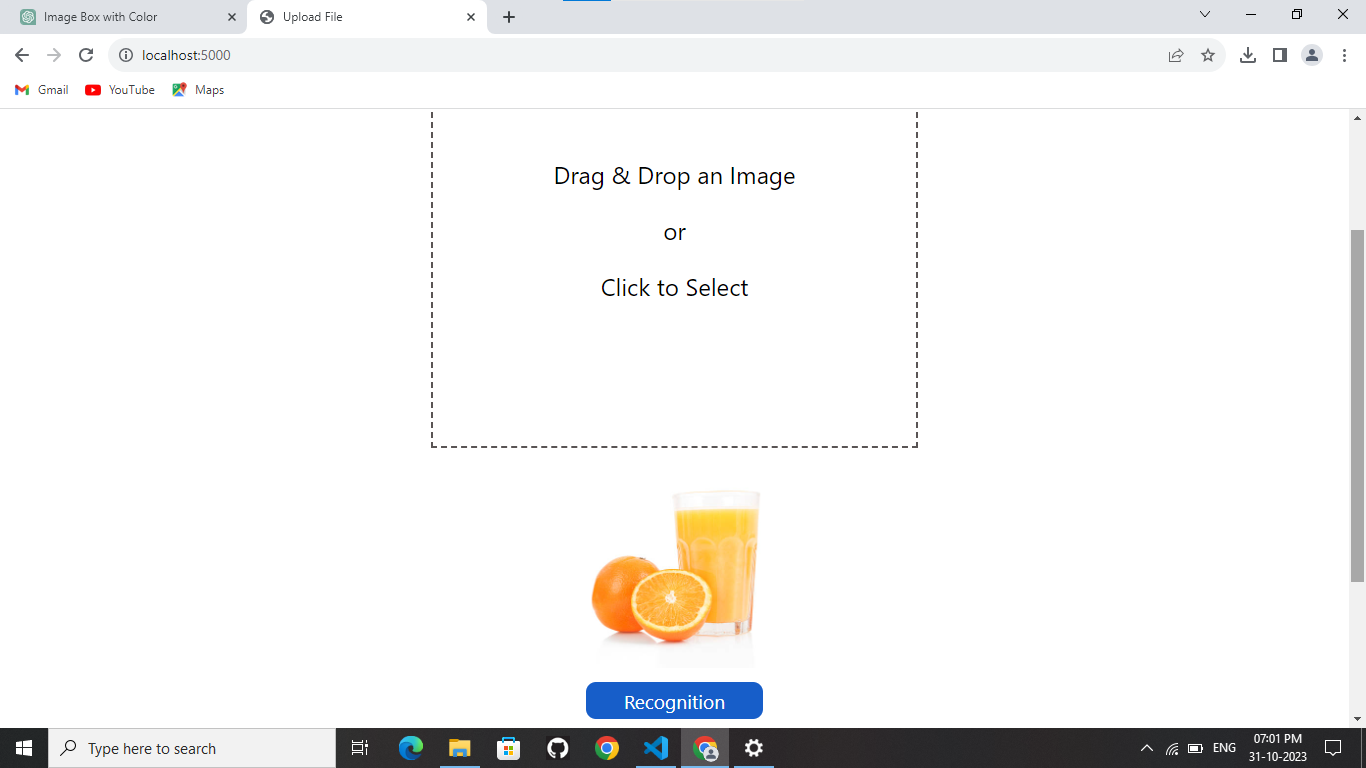
* Open a web browser.
* Enter the following URL: **http://localhost:5000/**

You should now see the user interface for the Image Recognition System.

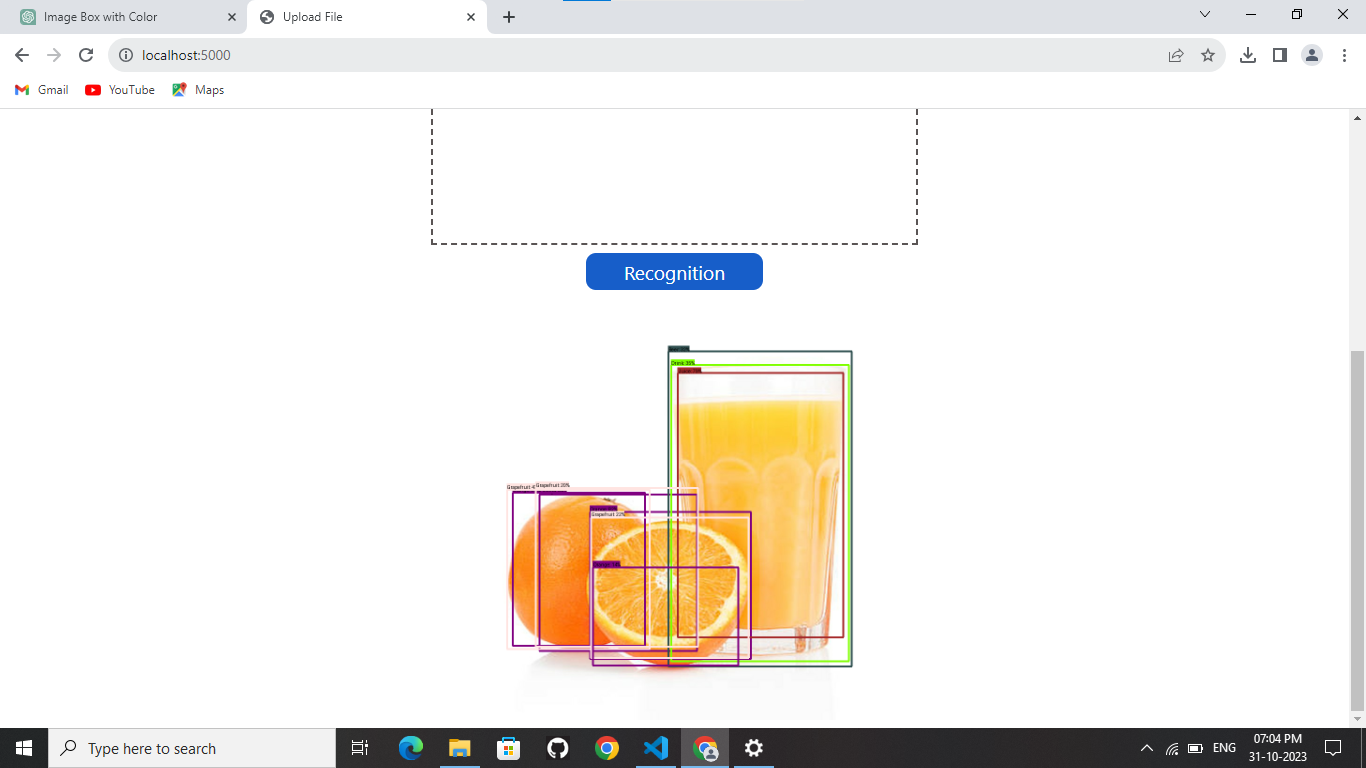
**5. Image Upload and Recognition**

* Click the image upload area to select an image from your local device or use drag-and-drop functionality to upload an image.
* The system will process the image and provide recognition results, including what the model recognized in the image and the associated accuracy level.

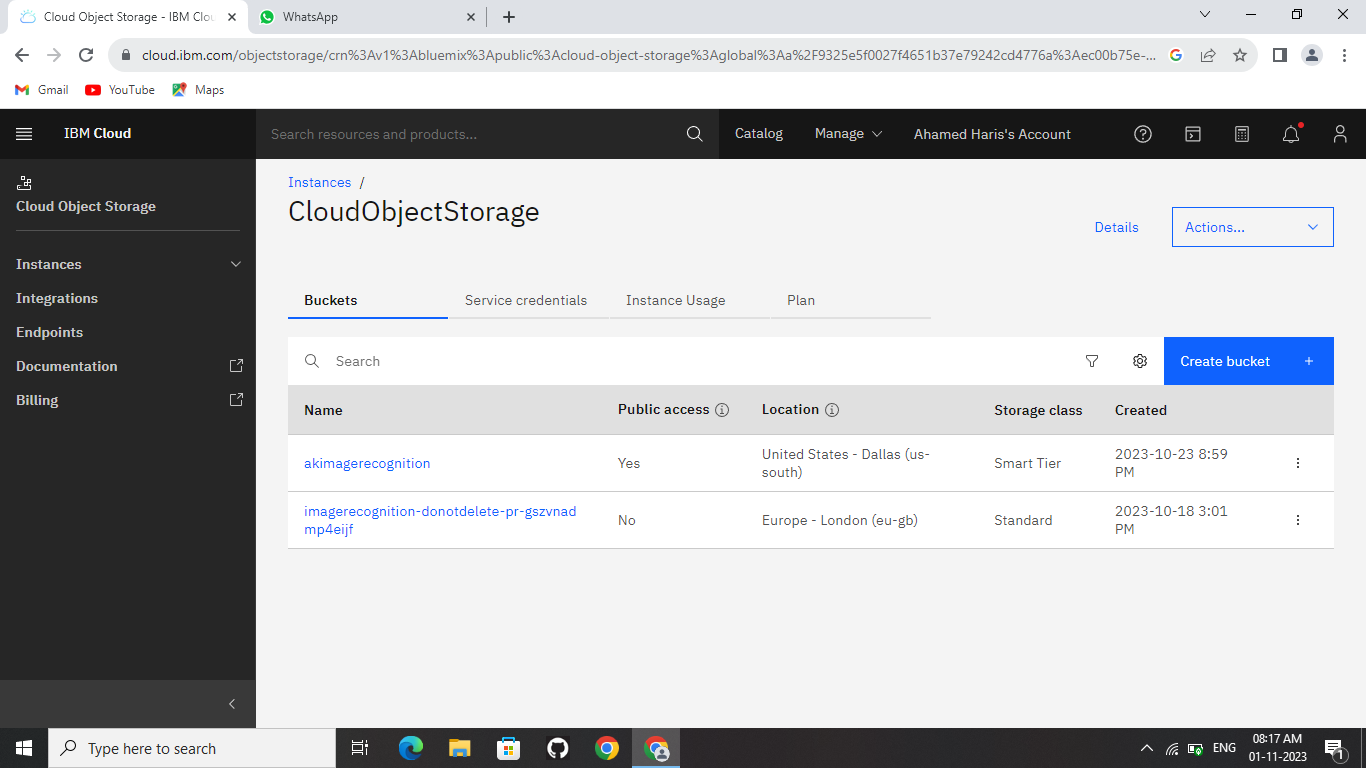
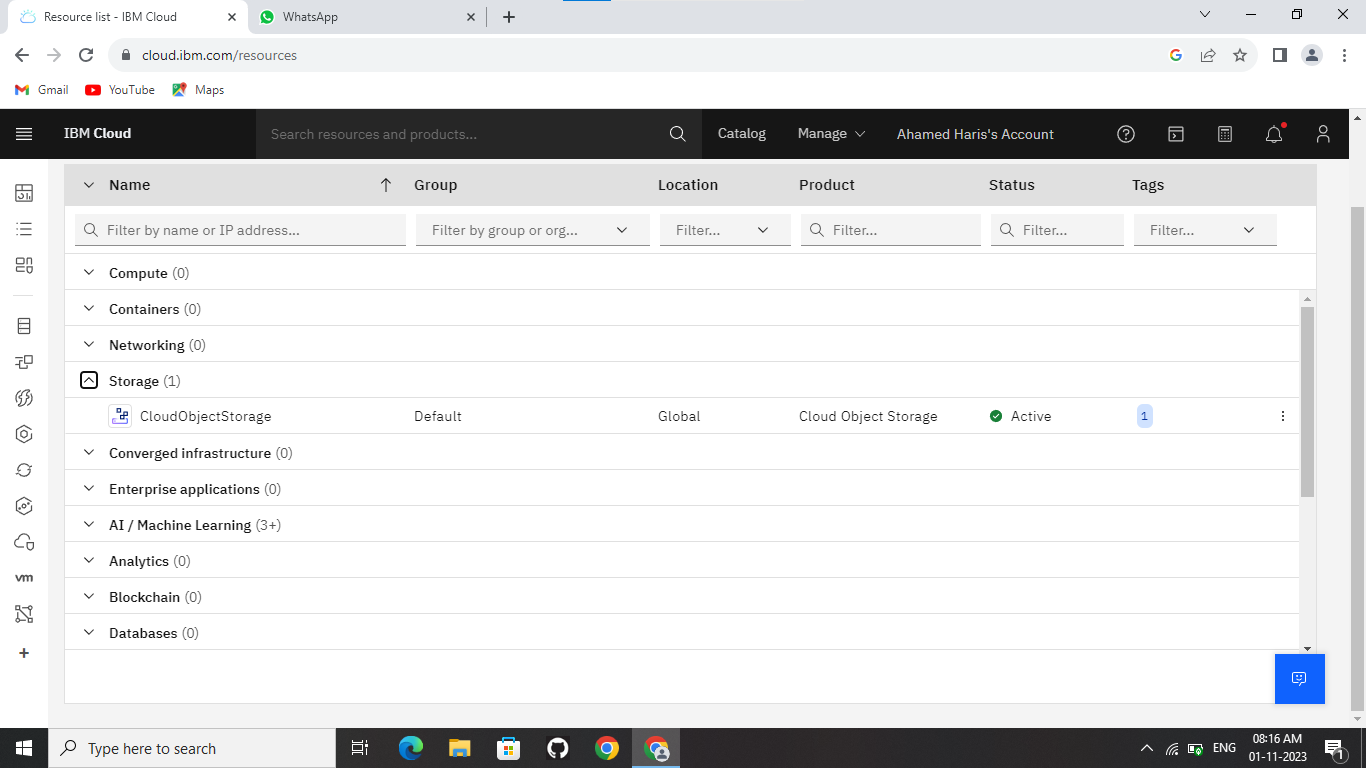
**OUTPUT:**



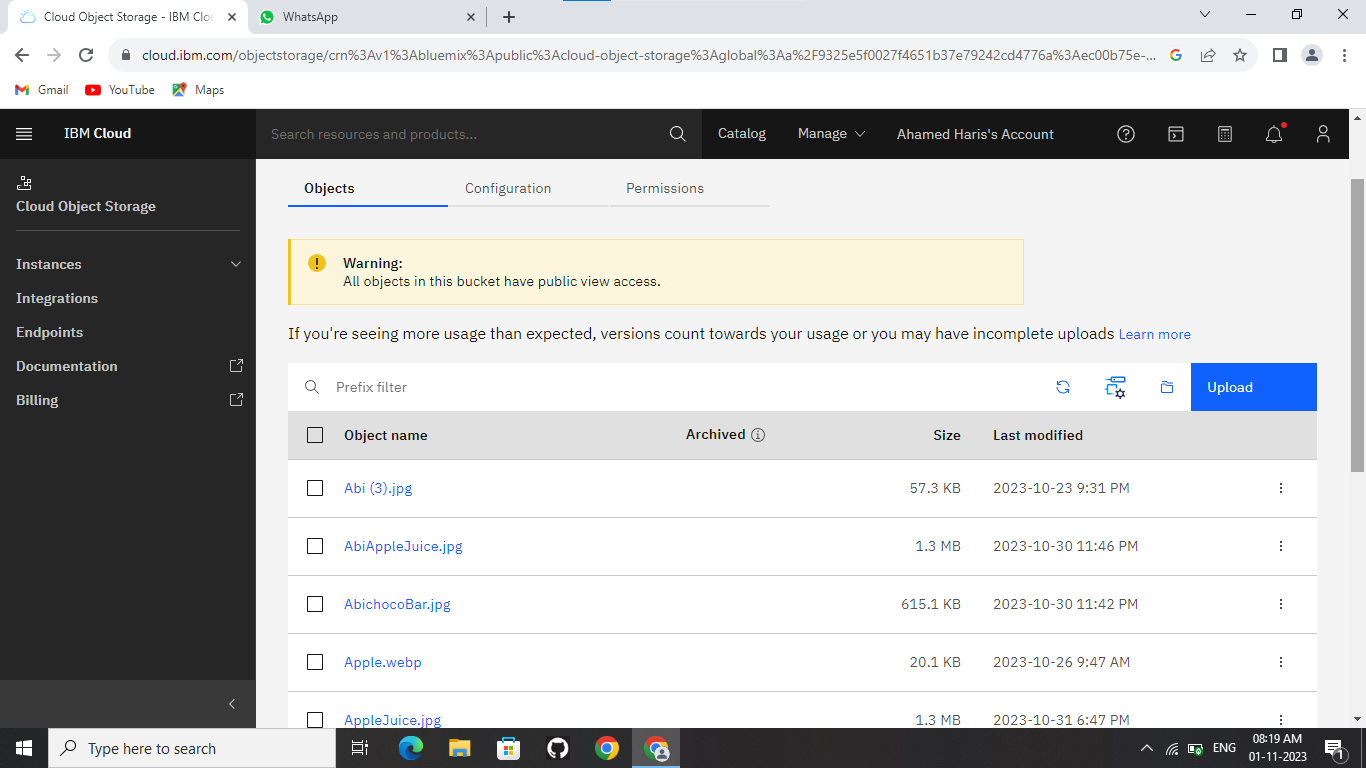
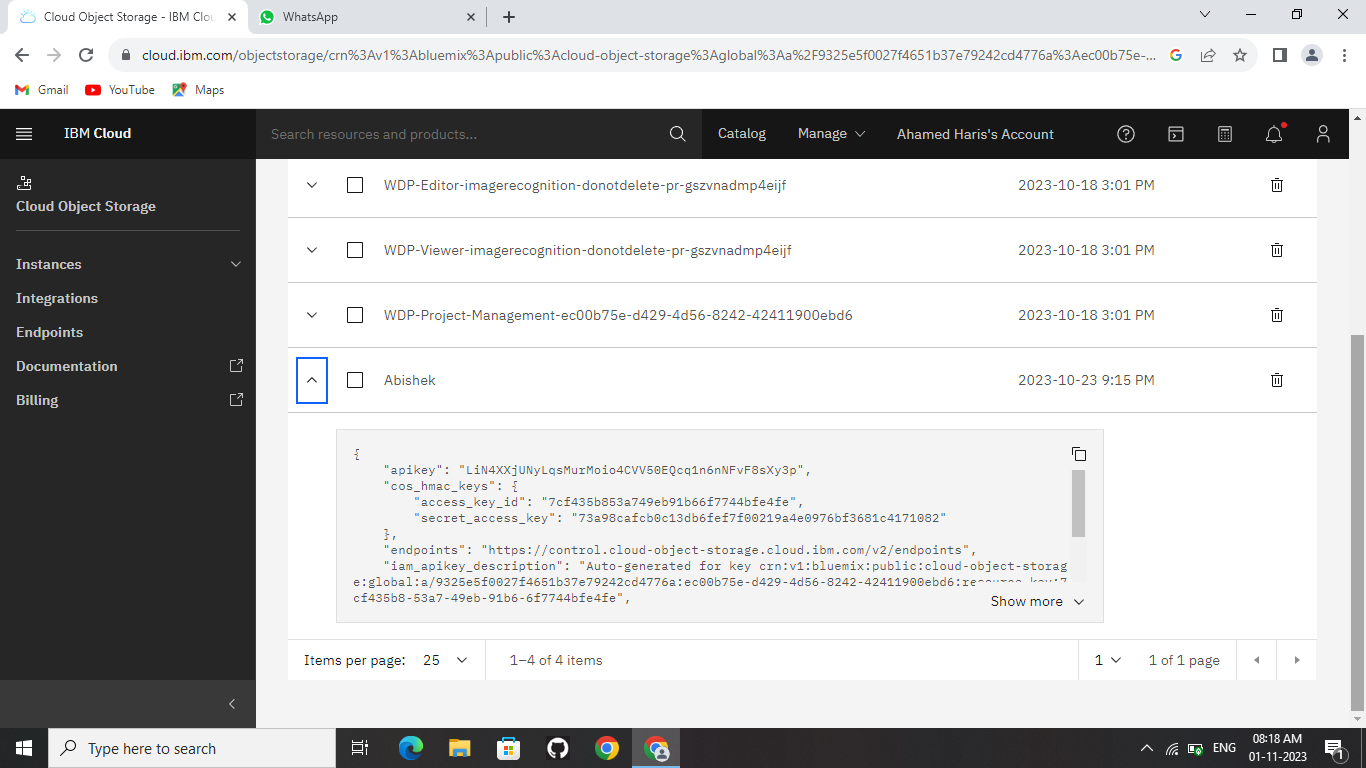
**Image upload on IBM object storage that image will recognize using tensorflow then again output of image also upload on IBM cloud object Storage using api.**



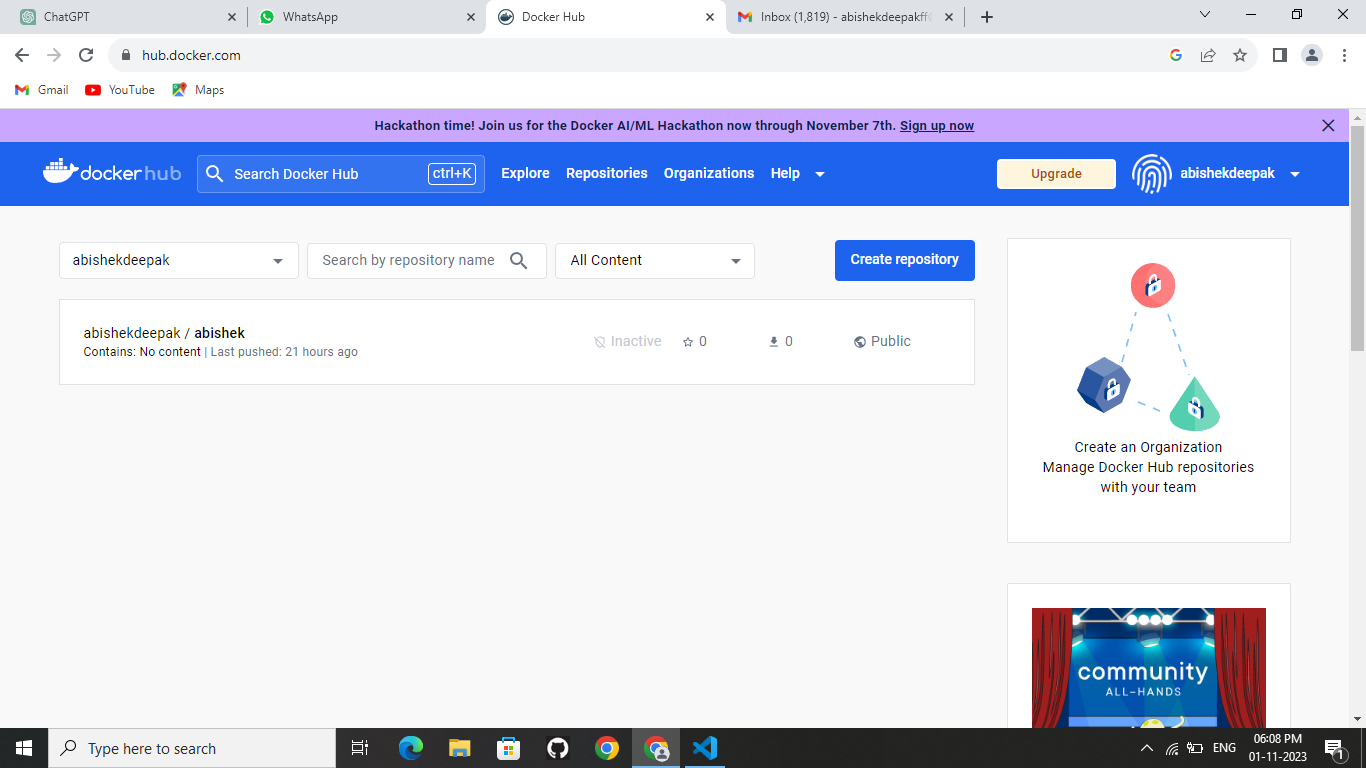
**IBM Cloud Object Storage:**



**IBM Cloud Object Storage:**



**Docker Hub:**



**Flask Program(app.py):**

from flask import Flask, request, render\_template

import tensorflow as tf

import tensorflow\_hub as hub

import matplotlib.pyplot as plt

import tempfile

from six.moves.urllib.request import urlopen

from six import BytesIO

import numpy as np

from PIL import Image

from PIL import ImageColor

from PIL import ImageDraw

from PIL import ImageFont

from PIL import ImageOps

import time

from ibm\_botocore.client import Config, ClientError

import ibm\_boto3

credentials = {

    'IBM\_API\_KEY\_ID': 'LiN4XXjUNyLqsMurMoio4CVV50EQcq1n6nNFvF8sXy3p',

    'IAM\_SERVICE\_ID': 'crn:v1:bluemix:public:cloud-object-storage:global:a/9325e5f0027f4651b37e79242cd4776a:ec00b75e-d429-4d56-8242-42411900ebd6:bucket:akimagerecognition',

    'ENDPOINT': 'https://s3.us-south.cloud-object-storage.appdomain.cloud',

    'IBM\_AUTH\_ENDPOINT':'https://iam.cloud.ibm.com/identity/token',

    'BUCKET': 'akimagerecognition',

    'FILE': 'chocoBar.jpg'

}

cos = ibm\_boto3.client(*service\_name*='s3',

*ibm\_api\_key\_id*=credentials['IBM\_API\_KEY\_ID'],

*ibm\_service\_instance\_id*=credentials['IAM\_SERVICE\_ID'],

*ibm\_auth\_endpoint*=credentials['IBM\_AUTH\_ENDPOINT'],

*config*=Config(*signature\_version*='oauth'),

    # config=Config(signature\_version='s3v4'),

*endpoint\_url*=credentials['ENDPOINT'])

*def* display\_image(*image*):

  # fig = plt.figure(figsize=(20, 15))

  plt.grid(False)

  plt.imshow(*image*)

*def* download\_and\_resize\_image(*url*, *new\_width*=256, *new\_height*=256,

*display*=False):

  \_, filename = tempfile.mkstemp(*suffix*=".jpg")

  response = urlopen(*url*)

  image\_data = response.read()

  image\_data = BytesIO(image\_data)

  pil\_image = Image.open(image\_data)

  pil\_image = ImageOps.fit(pil\_image, (*new\_width*, *new\_height*), Image.LANCZOS)

  pil\_image\_rgb = pil\_image.convert("RGB")

  pil\_image\_rgb.save(filename, *format*="JPEG", *quality*=90)

  print("Image downloaded to %s." % filename)

  return filename

*def* draw\_bounding\_box\_on\_image(*image*,

*ymin*,

*xmin*,

*ymax*,

*xmax*,

*color*,

*font*,

*thickness*=4,

*display\_str\_list*=()):

  """Adds a bounding box to an image."""

  draw = ImageDraw.Draw(*image*)

  im\_width, im\_height = *image*.size

  (left, right, top, bottom) = (*xmin* \* im\_width, *xmax* \* im\_width,

*ymin* \* im\_height, *ymax* \* im\_height)

  draw.line([(left, top), (left, bottom), (right, bottom), (right, top),

             (left, top)],

*width*=*thickness*,

*fill*=*color*)

  display\_str\_heights = [*font*.getbbox(ds)[3] for ds in *display\_str\_list*]

  total\_display\_str\_height = (1 + 2 \* 0.05) \* sum(display\_str\_heights)

  if top > total\_display\_str\_height:

    text\_bottom = top

  else:

    text\_bottom = top + total\_display\_str\_height

  for display\_str in *display\_str\_list*[::-1]:

    bbox = *font*.getbbox(display\_str)

    text\_width, text\_height = bbox[2], bbox[3]

    margin = np.ceil(0.05 \* text\_height)

    draw.rectangle([(left, text\_bottom - text\_height - 2 \* margin),

                    (left + text\_width, text\_bottom)],

*fill*=*color*)

    draw.text((left + margin, text\_bottom - text\_height - margin),

              display\_str,

*fill*="black",

*font*=*font*)

    text\_bottom -= text\_height - 2 \* margin

*def* draw\_boxes(*image*, *boxes*, *class\_names*, *scores*, *max\_boxes*=10, *min\_score*=0.1):

  """Overlay labeled boxes on an image with formatted scores and label names."""

  colors = list(ImageColor.colormap.values())

  try:

    font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSansNarrow-Regular.ttf",

                              25)

  except IOError:

    print("Font not found, using default font.")

    font = ImageFont.load\_default()

  for i in range(min(*boxes*.shape[0], *max\_boxes*)):

    if *scores*[i] >= *min\_score*:

      ymin, xmin, ymax, xmax = tuple(*boxes*[i])

      display\_str = "{}: {}%".format(*class\_names*[i].decode("ascii"),

                                     int(100 \* *scores*[i]))

      color = colors[hash(*class\_names*[i]) % len(colors)]

      image\_pil = Image.fromarray(np.uint8(*image*)).convert("RGB")

      draw\_bounding\_box\_on\_image(

          image\_pil,

          ymin,

          xmin,

          ymax,

          xmax,

          color,

          font,

*display\_str\_list*=[display\_str])

      np.copyto(*image*, np.array(image\_pil))

  return *image*

*def* load\_img(*path*):

  img = tf.io.read\_file(*path*)

  img = tf.image.decode\_jpeg(img, *channels*=3)

  return img

*def* run\_detector(*detector*, *path*,*fileimage*):

  img = load\_img(*path*)

  converted\_img  = tf.image.convert\_image\_dtype(img, tf.float32)[tf.newaxis, ...]

  start\_time = time.time()

  result = *detector*(converted\_img)

  end\_time = time.time()

  result = {key:value.numpy() for key,value in result.items()}

  print("Found %d objects." % len(result["detection\_scores"]))

  print("Inference time: ", end\_time-start\_time)

  image\_with\_boxes = draw\_boxes(

      img.numpy(), result["detection\_boxes"],

      result["detection\_class\_entities"], result["detection\_scores"])

  image\_with\_boxes\_pil = Image.fromarray(image\_with\_boxes)

  print('upload')

  with BytesIO() as output:

        image\_with\_boxes\_pil.save(output, *format*="PNG")

        output.seek(0)

        cos.upload\_fileobj(output,*Bucket*='akimagerecognition',*Key*=*fileimage*.filename)

module\_handle = "https://tfhub.dev/google/faster\_rcnn/openimages\_v4/inception\_resnet\_v2/1"

detector = hub.load(module\_handle).signatures['default']

app = Flask(\_\_name\_\_)

@app.route('/', *methods*=['GET', 'POST'])

*def* index():

    if request.method == 'POST':

        file = request.files['file']

        cos.upload\_fileobj(file,*Bucket*='akimagerecognition',*Key*=file.filename)

        image\_urlnew="https://akimagerecognition.s3.us-south.cloud-object-storage.appdomain.cloud/"+file.filename

        downloaded\_image\_path = download\_and\_resize\_image(image\_urlnew, 1280, 856, True)

        run\_detector(detector, downloaded\_image\_path,file)

        return render\_template('index.html',*image\_url*=image\_urlnew)

    return render\_template('index.html',*image\_url*='')

if \_\_name\_\_ == '\_\_main\_\_':

    print('Hello')

    app.run(*debug*=False)

**Templates/index.html:**

<!DOCTYPE html>

<html>

<head>

    <title>Upload File</title>

    <style>

        \*{

*font-family*: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

        }

        .image-box {

*max-width*: 600px;

*max-height*: 500px;

*min-height*: 200px;

*min-width*: 300px;

*background-color*: #ffffff; /\* Set the background color you want \*/

*text-align*:center;

*display*: flex;

*flex-direction*: column;

*align-items*: center;

*justify-content*: center;

*margin*: 0 auto;

        }

        .image-box img {

*max-width*: 100%; /\* Ensure the image doesn't exceed the container \*/

*max-height*: 100%; /\* Ensure the image doesn't exceed the container \*/

        }

        .drop-container {

*display*: flex;

*flex-direction*: column;

*align-items*: center;

*margin*: 20px 0;

*min-width*:100%;

*font-size*: 1.5em;

}

.drop-area {

    /\* #3498db \*/

*border*: 2px dashed rgb(90, 85, 85);

*text-align*: center;

*cursor*: pointer;

*max-width*: 400px;

*padding*: 5em 5em;

}

.image-preview {

*display*: none;

*margin-top*: 20px;

}

#previewImage {

*max-width*: 100%;

*max-height*: 200px;

}

#subBtn{

*color*: #ffffff;

*background-color*: rgb(23, 94, 201);

*padding*: 0.3em 2em;

*font-size*: 0.8em;

*outline*: none;

*border*: none;

*border-radius*: 10px;

*margin*: 8px;

}

    </style>

</head>

<body>

    <center><h1>Image Recognition</h1></center>

    <form action="/" method="POST" enctype="multipart/form-data">

    <div class="drop-container">

        <input type="file" id="fileInput" name="file" accept="image/\*" style="display: none;" required>

        <label for="fileInput" class="drop-area">

            <p>Drag & Drop an Image</p>

            <p>or</p>

            <p>Click to Select</p>

        </label>

        <div class="image-preview">

            <img id="previewImage" src="" alt="Image Preview">

        </div>

        <input type="submit" value="Recognize" id="subBtn">

    </form>

    </div>

    <div class="containerBox">

    <div class="image-box">

        <img src="{{ image\_url }}"  onerror=hideImage() id="opImage">

    </div>

    </div>

</body>

<script>

*const* fileInput = document.getElementById('fileInput');

*const* dropArea = document.querySelector('.drop-area');

*const* imagePreview = document.querySelector('.image-preview');

*const* previewImage = document.getElementById('previewImage');

fileInput.addEventListener('change', *function* () {

*const* file = fileInput.files[0];

    if (file) {

        displayImage(file);

    }

});

dropArea.addEventListener('dragover', *function* (*e*) {

*e*.preventDefault();

    dropArea.classList.add('drag-over');

});

dropArea.addEventListener('dragleave', *function* () {

    dropArea.classList.remove('drag-over');

});

dropArea.addEventListener('drop', *function* (*e*) {

*e*.preventDefault();

    dropArea.classList.remove('drag-over');

*const* file = *e*.dataTransfer.files[0];

    if (file) {

        displayImage(file);

    }

});

*function* hideImage(){

document.getElementById('opImage').style.display='none';

}

*function* displayImage(*file*) {

    if (*file*.type.startsWith('image/')) {

*const* reader = new *FileReader*();

        reader.onload = *function* (*e*) {

            imagePreview.style.display = 'block';

            previewImage.src = *e*.target.result;

        };

        reader.readAsDataURL(*file*);

    } else {

        alert('Please select an image file.');

    }

}

</script>

</html>