KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG - UIT

BÁO CÁO THỰC HÀNH

Lab 6

CÔNG NGHỆ INTERNET OF THINGS HIỆN ĐẠI

Hoàn thiện giải pháp IoT

GVHD: **Phan Trung Phát**

Lớp: **NT532.O21.MMCL.2**

Họ và tên	MSSV
Tống Võ Anh Thuận	21522652
Trịnh Vinh Đại	21521915
Lê Huỳnh Quang Vũ	21522797

ĐÁNH GIÁ KHÁC (*):

Nội dung	Kết quả
Tổng thời gian thực hiện bài thực hành trung bình (1)	6 ngày
Link Video thực hiện (2) *(nếu có)*	Video n Code
Ý kiến (3) *(nếu có)* + Khó khăn + Đề xuất ...	
Điểm tự đánh giá (4)	10/10

(*): phần (1) và (4) bắt buộc thực hiện.

Phần bên dưới là báo cáo chi tiết của nhóm/cá nhân thực hiện.

WARNING LƯU HÀNH NỘI BỘ

Câu hỏi 1. Thu thập hình ảnh các thành viên trong nhóm, sau đó huấn luyện lại mô hình Facenet với độ chính xác trên 60%. (Sử dụng mã nguồn cung cấp trong bài thực hành số 5).

Since our model from Lab05 is good enough to be reused for this lab, no further explanation about collecting data or model training will be written here.

Câu hỏi 2. Sử dụng Apache Kafka làm kênh giao tiếp giữa Client và Edge / Cloud Server. Cụ thể, Client sử Kafka Producer để gửi hình ảnh tới Topic trên Kafka Broker. Lúc đó, Edge / Cloud Server sử dụng Kafka Consumer để nhận hình ảnh đó. Hình ảnh đưa vào mô hình FaceNet để tiến hành nhận diện. Kết quả được Edge / Cloud Server trả về Client cũng thông qua quá trình pub/sub tương tự như trên. Client nhận kết quả và xuất ra màn hình.

1. Setup:

Components in use:

- 2 laptops: 1 Edge Server, 1 Kafka Broker
- 1 Jetson Nano which acts as Client
- 1 IMX-219 160 CSI camera

Responsibility:

- Kafka Broker:
  - Configured *docker-compose.yml* and *config.py* as localhost
  - Built and run Kafka image
  - Created 2 Kafka topics: *send_image* and *receive_result*
  - Published IP address for Server and Client configuration.

  Client:
  - Configured *docker-compose.yml* and *config.py* due to Broker IP address
  - Built and run Kafka image
  - Captured image and sent to Edge Server on *send_image* topic
  - Received result from Egde Server on *receive_result* topic
- Edge Server:
  - Reuse the flask server code from Lab05 to load the classifier model.
  - Add in the Kafka producer and consumer logic for both topics but reverse the role.

Figure 1. Jetson Nano.

2. Code explanation:

a) Kafka Broker.

In *docker-compose.yml*, there are 2 services: **Zookeeper** and **Kafka** and the image provider is **Bitnami.**

```
zookeeper:
  image: "bitnami/zookeeper:latest"
  container_name: zookeeper
  ports:
    - "2181:2181"
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes
```

Zookeeper uses Bitnami latest image version. Container's name for zookeeper is "zoopkeeper". This image maps port 2181 in host to port 2181 in container. It allows anonymous login.

```
kafka:
  image: "bitnami/kafka:latest"
  container_name: kafka
  user: root
  ports:
    - "9092:9092"
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_LISTENERS=PLAINTEXT://:9092
    - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092
    - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
    - ALLOW_PLAINTEXT_LISTENER=yes
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  depends_on:
    - zookeeper
```

Like Zookeeper, Kafka also uses latest version from Bitnami. This image runs as root user, maps port 9092 from host to container. For the environment, Broker ID is 1, Advertised listeners is localhost:9092, Zookeeper port is 2181 and allow

plaintext listener. PLAINTEXT in environment means the listener will be without authentication and non-encrypted. Finally, this image depends on Zookeeper.

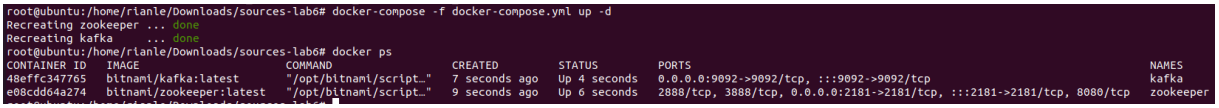


Figure 2. Kafka Broker.

b) Client.

```
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_LISTENERS=PLAINTEXT://:9092
    - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://<BROKER_IP>:9092
    - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
    - ALLOW_PLAINTEXT_LISTENER=yes
```

For *docker-compose.yml*, we change listener's IP address from localhost to Broker IP address and keep the other configuration.

```
kafka_ip = "<BROKER_IP>:9092"
```

We also replace default IP address with Broker IP address in *config.py* file.

Due to the replacement of hardware, we must make some modifications to *send_image.py* file:

```
def gstreamer_pipeline(
    capture_width=1280,
    capture_height=720,
    display_width=1280,
    display_height=720,
    framerate=20,  # Set framerate to 20 fps
    flip_method=0,
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, "
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )
```

This function creates a gst stream for CSI camera on Jetson Nano.

```
topic_name = "send_image"
```

```
p = KafkaProducer(
    bootstrap_servers=[config.kafka_ip],
    max_request_size = 9000000,
)
```

This code defined Kafka topic name and producer.

```
cam = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
start_time = time.time()
duration = 10  # capture duration in seconds

while True:
    ret, frame = cam.read()

    # Display the frame (optional)
    cv2.imshow('CSI Camera', frame)

    # Check if the duration has passed
    if time.time() - start_time > duration:
        frame = cv2.resize (frame, dsize=None, fx=0.2, fy=0.2)
        ret, buffer = cv2.imencode('.jpg', frame)
        p.send(topic_name, buffer.tobytes())
        p.flush()
        print("Sent!")
        start_time = time.time()

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

This code creates a camera stream. Within 10 seconds, it captures images and displays them in a new window. The last image will be resized and sent to Kafka Producer on *send_image* topic when it reaches duration.

In *recv_result.py* file, we subscribe to *receive_result* topic to get message from Server.

```
import json
from kafka import KafkaConsumer
import config
topic_name = "receive_result"
c = KafkaConsumer(
    topic_name,
    bootstrap_servers = [config.kafka_ip],
    auto_offset_reset = 'latest',
    enable_auto_commit = True
)

for message in c:
    print("Message: ", message.value)
```

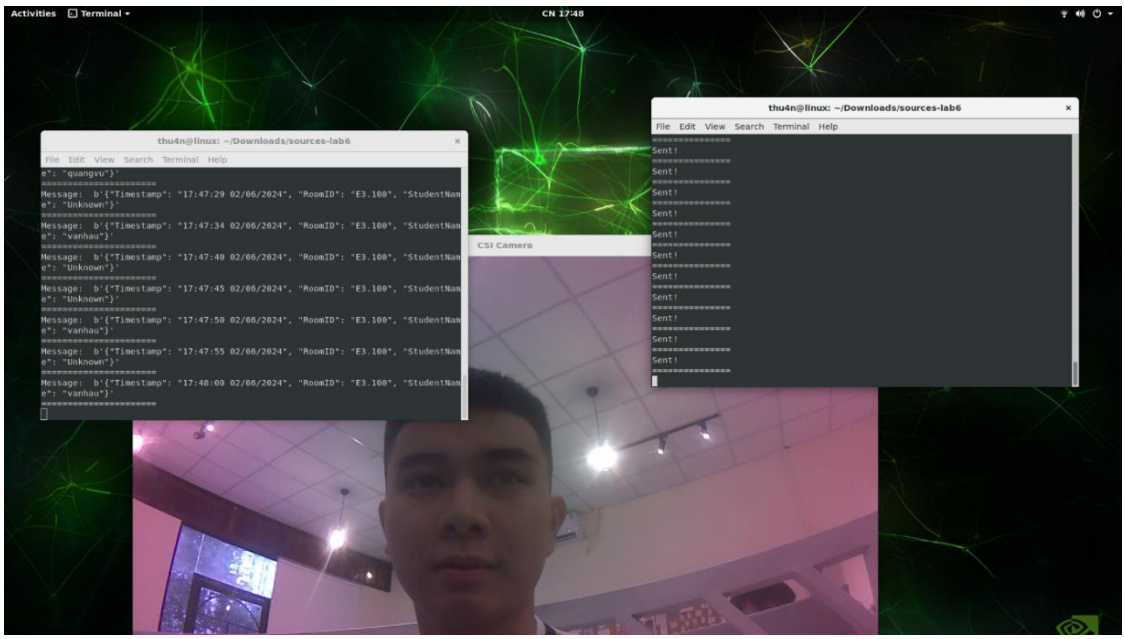


Figure 3. Jetson Nano as Client.

c) Edge server:

```
with tf.Graph().as_default():

    # Cai dat GPU neu co
    gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.6)
    sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(gpu_options=gpu_options, log_device_placement=False))

    with sess.as_default():
        # Load the model
        print('Loading feature extraction model')
        facenet.load_model(FACENET_MODEL_PATH)

        images_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("input:0")
        embeddings = tf.compat.v1.get_default_graph().get_tensor_by_name("embeddings:0")
        phase_train_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("phase_train:0")
        embedding_size = embeddings.get_shape()[1]

        pnet, rnet, onet = align.detect_face.create_mtcnn(sess, "align")
```

The above piece of code as well as the model inference logic is reused from Lab05 so no further explanation is needed here.

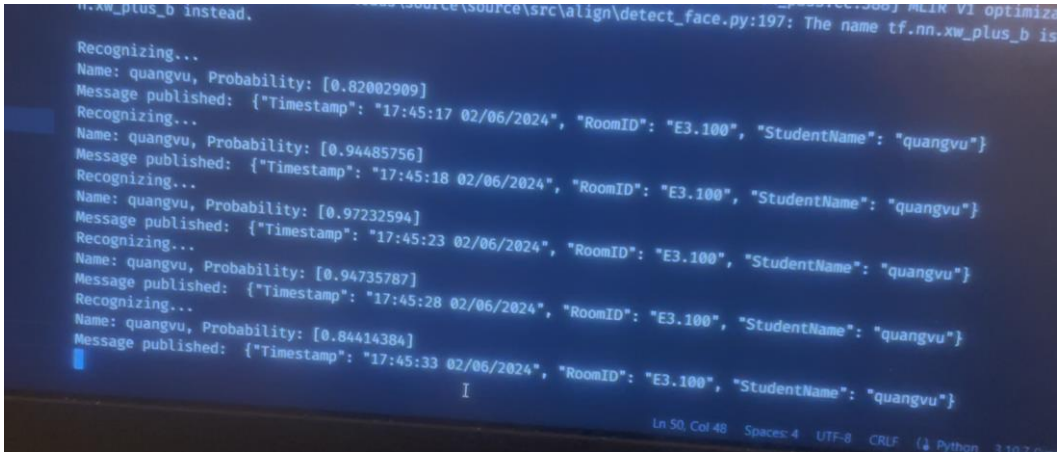
```
# Kafka config
sub_topic = "send_image"
pub_topic = "receive_result"

c = KafkaConsumer(
    sub_topic,
    bootstrap_servers = ["192.168.0.82:9092"],
    auto_offset_reset = 'latest',
    enable_auto_commit = True,
    fetch_max_bytes = 9000000,
    fetch_max_wait_ms = 10000,
)

p = KafkaProducer(
    bootstrap_servers=["192.168.0.82:9092"],
    max_request_size = 9000000,
)
```

This part is to set the parameters for the Kafka Consumer and Producer where it will receive messages from “send_image” and then publish to “receive_result” using the local IP address of the Kafka broker.

```
for message in c:
    stream = message.value
    stream = np.frombuffer(stream, dtype=np.uint8)
    image = cv2.imdecode(stream, cv2.IMREAD_COLOR)
    print("Recognizing ... ")
    prediction = model_inference(image)
    now = datetime.now()
    formatted_time = now.strftime("%H:%M:%S %d/%m/%Y")
    json_msg = json.dumps({"Timestamp":formatted_time,"RoomID":"E3.100","StudentName":prediction})
    p.send(pub_topic, json_msg.encode("utf-8"))
    client.publish('v1/devices/me/telemetry', json_msg, 1)
    print("Message published: ", json_msg)
    p.flush()
```

The above code is to read the value of each message it receives and then run cv2.imdecode to get the image. This image will then be passed as an argument for the model inference function. After prediction is made, additional information is added before it is published to the “receive_result” topic.

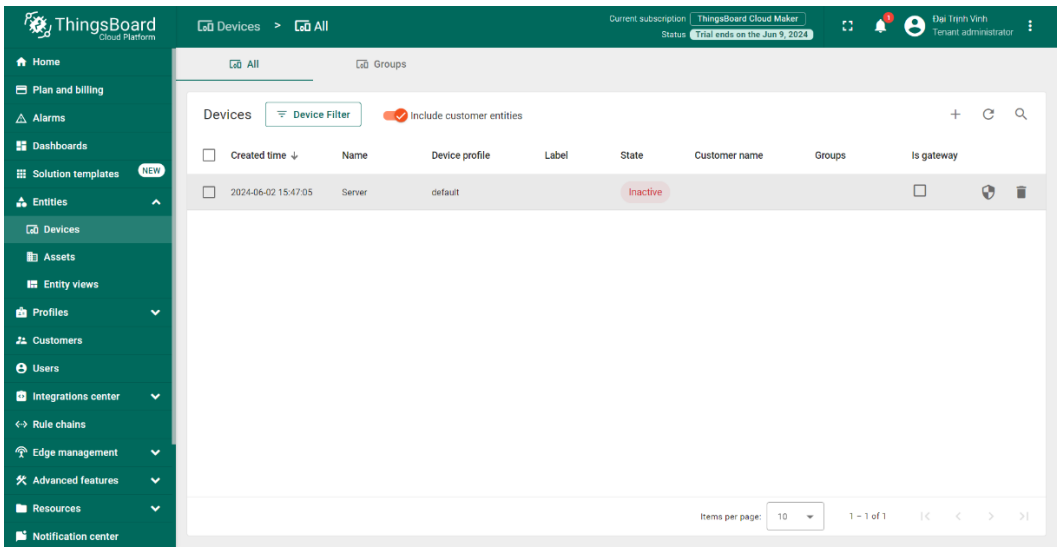
Final result (cropped from video):



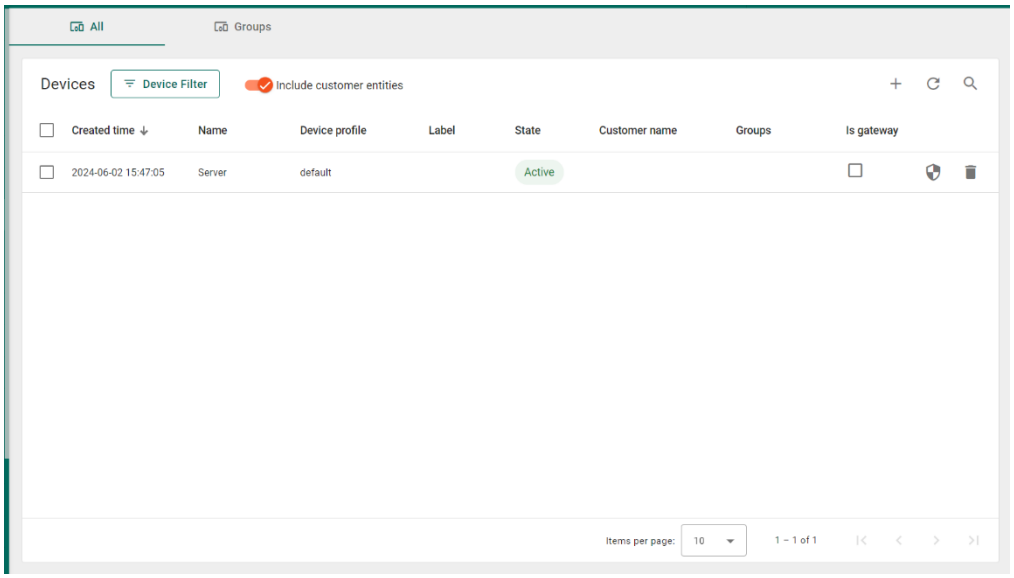
Câu hỏi 3. Kết quả nhận diện đồng thời được gửi lên IoT Platform để lưu trữ quản lý. Yêu cầu lưu trữ các thông tin như: Timestamp (h:m:s d/m/y), Room ID, Student Name.

The IoT platform we will be using for this lab is **ThingsBoard Cloud**. It is free and open source (it is also recommended by the instructor). Documentation is available here: https://thingsboard.io/docs/paas/

My device name is “Server”. When no requests have been sent, the device's status is "inactive".



If an MQTT messge has been sent using the access token provided when creating the device, it will appear as "active" on the platform.



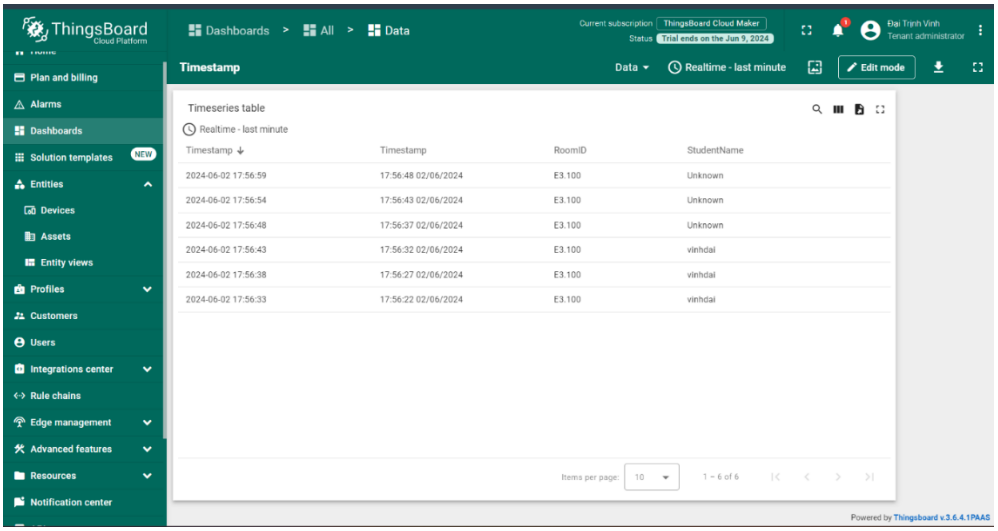
Code explaination (same file as edge server):

```
47    # MQTT config
48    def on_connect(client, userdata, flags, rc):
49        print("Connected with result code " + str(rc))
50        client.subscribe('v1/devices/me/telemetry')
51
52    def on_message(client, userdata, msg):
53        print(msg.topic + " " + str(msg.payload))
54
55    THINGSBOARD_HOST = 'mqtt.thingsboard.cloud'
56    ACCESS_TOKEN = 'Q24KljOeksGaq2fgUipo'
57
58    client = mqtt.Client()
59    client.on_connect = on_connect
60    client.on_message = on_message
61
62    client.username_pw_set(ACCESS_TOKEN)
63
64    client.connect(THINGSBOARD_HOST, 1883, 60)
```

This code connects an MQTT client to ThingsBoard, declaring the topic, access token, and connection port 1883. We test the device activation by running this code.

```
135     for message in c:
136         stream = message.value
137         stream = np.frombuffer(stream, dtype=np.uint8)
138         image = cv2.imdecode(stream, cv2.IMREAD_COLOR)
139         print("Recognizing...")
140         prediction = model_inference(image)
141         now = datetime.now()
142         formatted_time = now.strftime("%H:%M:%S %d/%m/%Y")
143         json_msg = json.dumps({"Timestamp":formatted_time,"RoomID":"E3.100","StudentName":prediction})
144         p.send(pub_topic, json_msg.encode("utf-8"))
145         client.publish('v1/devices/me/telemetry', json_msg, 1)
146         print("Message published: ", json_msg)
147         p.flush()
```

The code on line 145 sends real-time results to Thingsboard, properties include Timestamp, RoomID, and StudentName.
And below are the results received on Thingsboard's dashboard with a timeseries table widget:



YÊU CẦU CHUNG

1) Đánh giá

- Chuẩn bị tốt các yêu cầu đặt ra trong bài thực hành.
- Sinh viên hiểu và tự thực hiện được bài thực hành, trả lời đầy đủ các yêu cầu đặt ra.
- Nộp báo cáo kết quả chi tiết những đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả *(nếu có)*; giải thích cho quan sát *(nếu có)*.
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

2) Báo cáo

- File .PDF hoặc .docx. Tập trung vào nội dung, giải thích.
- Nội dung trình bày bằng Font chữ Cambria hoặc Times New Roman (*tuy nhiên, phải chuyển đổi hết báo cáo này sang 1 font chữ thống nhất*) – cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: Mã lớp-LabX_MSSV1_MSSV2. (trong đó X là Thứ tự buổi Thực hành).

  Ví dụ: NT532.O21.1-Lab01_25520001_25520002
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài thực hành.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT