将 OpenVINO™ 推理结果通过 MQTT 推送给 EdgeX Foundry

张华乔(VMware | Senior Engineer) 和 张晶 合著

目 录

将 OpenVINO™ 推理结果通过 MQTT 推送给 EdgeX Foundry

1.1 常见的 EdgeAI 应用程序框架

EdgeAI 即常说的边缘智能，边缘智能包含两大核心功能：

1. **边缘**负责将现实世界的各种物理量数字化，然后整合数据，然后跟云端双向通信，例如，将温度、压力、位置等数字化后，整合信息上传并获得下行的命令
2. **智能**负责从原始的数据中抽取出有用的信息，例如，物体在视频中的位置和类别

边缘智能应用程序中常见的模块是 EdgeX Foundry 和 OpenVINO™ 工具包：

- **EdgeX Foundry** 负责从边缘处的传感器（即“物”物）收集数据，并作为双向传输引擎向企业、云和本地应用发送数据，以及从这些应用接收数据。
- **OpenVINO™** 工具包负责在边缘端实现 AI 模型的推理计算，获得推理结果

EdgeX Foundry 与 OpenVINO™ 工具包共同实现边缘智能的典型框架，如图 1-1 所示。

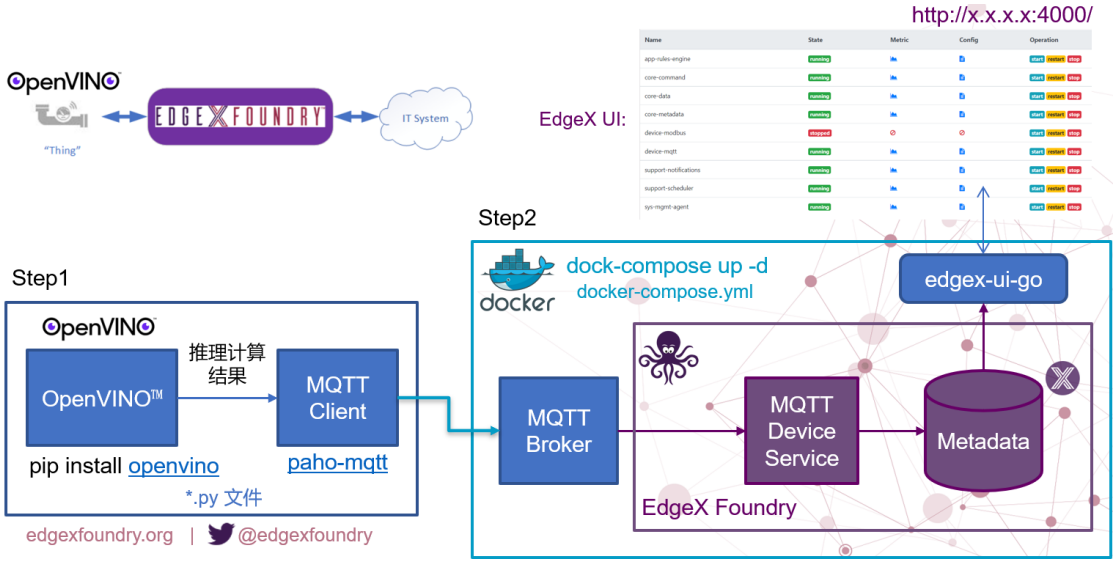


图 1-1　边缘智能应用程序典型框架

1.1.1 EdgeX Foundry 简介

EdgeX Foundry 是 LF Edge 旗下的一款开源、不受供应商限制的边缘物联网中间件平台，负责从边缘处的传感器（即“物”物）收集数据，并作为双向传输引擎向企业、云和本地应用发送数据，以及从这些应用接收数据，如图 1-2 所示。

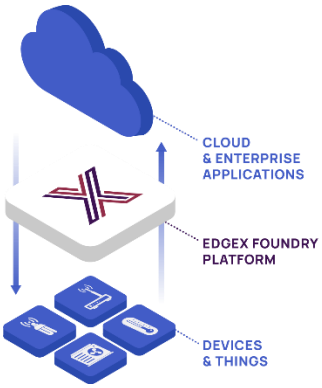


图 1-2　EdgeX Foundry

在边缘智能应用中使用 EdgeX Foundry 的好处：

- EdgeX 为设备数据引入、规范化及边缘智能 (AI/ML)提供可替换的参考服务
- EdgeX 共享支持新型物联网数据服务和高级边缘计算应用
- EdgeX 能加快完整边缘解决方案和/或边缘硬件解决方案的上市速度

1.1.2 OpenVINO™ 2022.1 简介

OpenVINO™ 工具包开源且商用免费，用于深度学习模型优化和部署，如图 1-3 所示。

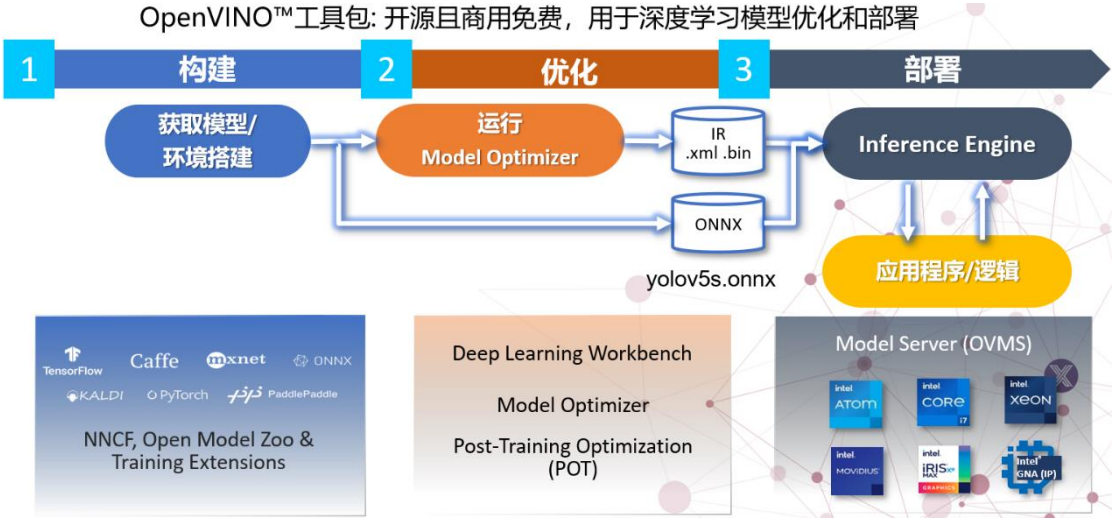


图 1-3　OpenVINO™ 工具包

OpenVINO™ 工具包 2022.1 版于 2022 年 3 月 22 日正式发布，根据官宣《OpenVINO™ 迎来迄今为止最重大更新，2022.1 新特性抢先看》，OpenVINO™ 2022.1 将是迄今为止最大变化的版本。

从开发者的角度来看，对于提升开发效率或运行效率有用的特性有：

- 提供**预处理 API 函数**。OpenVINO™ 2022.1 之前版本不提供 OpenVINO runtime 原生的用于数据预处理的 API 函数，开发者必须通过第三方库，比如，OpenCV，来实现数据预处理。OpenVINO™ 2022.1 自带的预处理 API 可以将所有预处理步

骤都集成到在执行图中，这样 iGPU、CPU、VPU 或今后 Intel 的独立显卡都能进行数据预处理，大大提高了执行效率；相比之前，用 OpenCV 实现的预处理，则只能在 CPU 上执行，如图 1-4 所示。

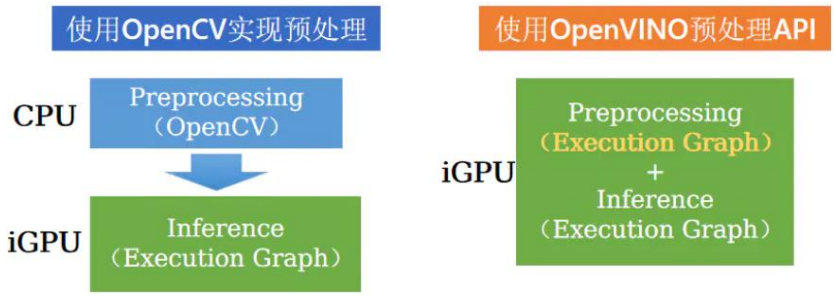


图 1-4　OpenVINO 预处理 API

- **ONNX 前端 API**。前端 API 意味着模型可以直接被 OpenVINO 读入，而无需使用模型优化器进行模型转换，这对于使用 ONNX 模型的开发人员非常有用。OpenVINO™ 2022.1 自动转换 ONNX 模型的速度(1.5 秒以内)相比之前版本，有极大提升，开发人员已经感受不到 ONNX 模型自动转换的时间消耗了。
- **AUTO 设备插件**。AUTO 设备插件自动将 AI 推理计算加载到最合适的硬件设备上(CPU, GPU, VPU 等)，无需额外的开发工作即可提高模型在异构系统中(例如：12 代 CPU + Iris Xe 集成显卡 + DG2 独显) 的推理性能和可移植性。
- 支持直接读入飞桨模型，如图 1-5 所示。

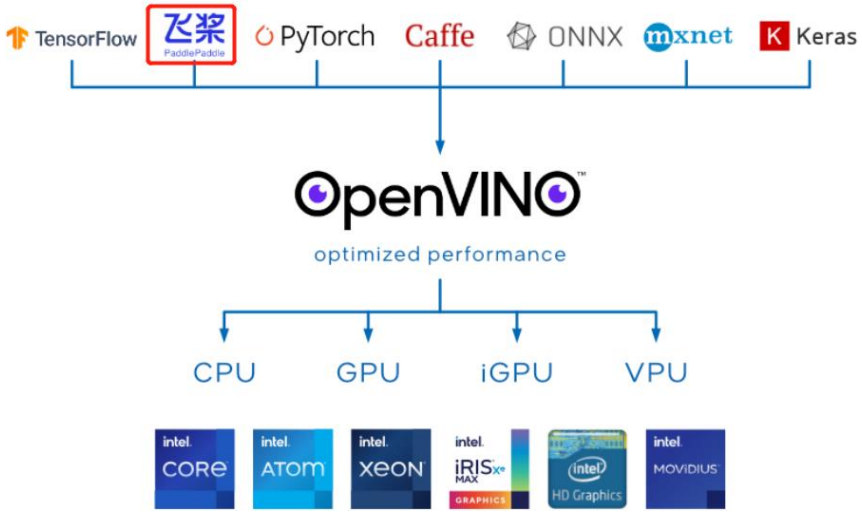


图 1-5　OpenVINO™ 支持的深度学习框架

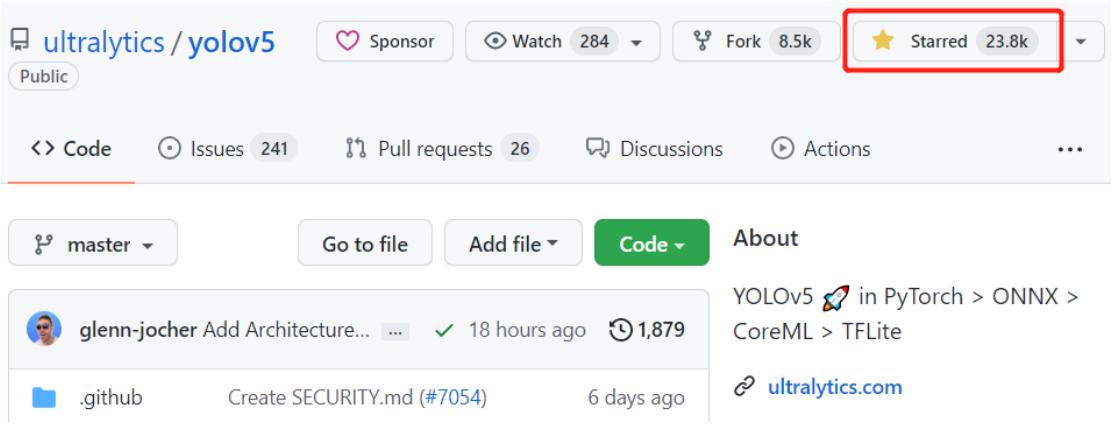
为实现将 OpenVINO™推理结果通过 MQTT 推送给 EdgeX Foundry，本文将依次介绍：

1. 使用 OpenVINO™ 开发 YOLOv5 模型推理程序
2. 运行带 ds-mqtt 的 EdgeX Foundry
3. 在 EdgeX UI 中新建 MQTT Device
4. 将 OpenVINO™推理结果推送给 MQTT Broker

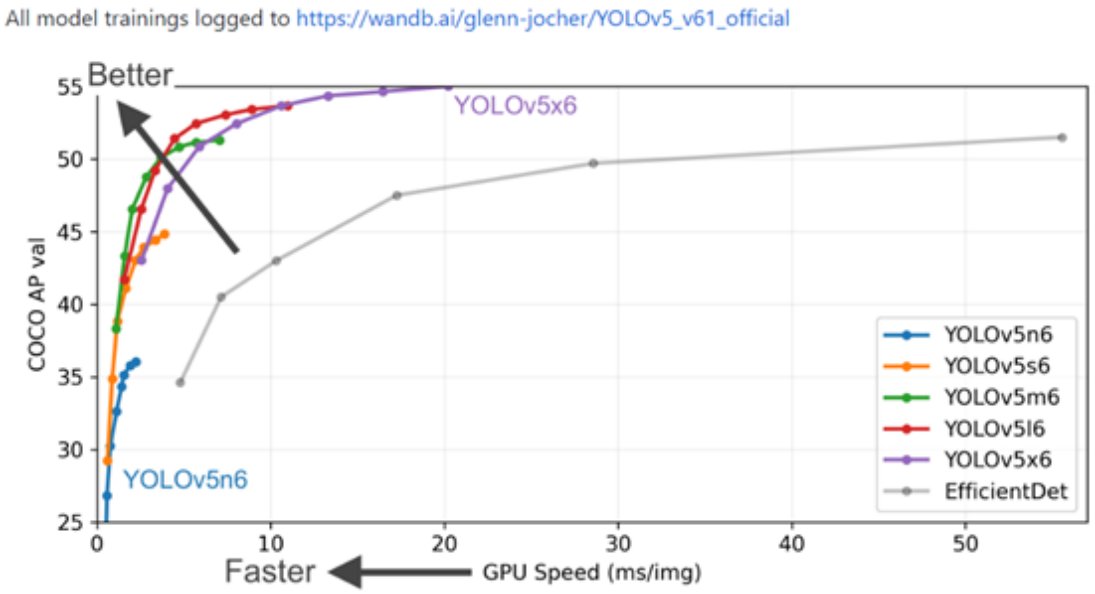
1.2 使用 OpenVINO™ 开发 YOLOv5 模型推理程序

1.2.1 YOLOv5 简介

Ultralytics 公司贡献的 YOLOv5 PyTorch (https://github.com/ultralytics/yolov5)实现版，由于其工程化和文档做的特别好，深受广大 AI 开发者的喜爱，GitHub 上的星标超过了 23.8K，而且被 PyTorch 官方收录于 PyTorch 的官方模型仓。在产业实践中，也有无数的开发者将 YOLOv5 直接用于自己的项目。



由于 YOLOv5 精度高速度快(最新的 YOLOv5 模型精度如下所示)，使得产业界里面使用 YOLOv5 模型做产品和项目的人非常多。



另外，YOLOv5 的工程化和文档化做的极好，没有任何基础的人，都可以在不到半天的时间内完成 YOLOv5 的开发环境搭建、模型训练、ONNX 模型导出。

YOLOv5 的文档链接：https://docs.ultralytics.com/

1.2.2 安装 YOLOv5 的 OpenVINO™推理程序开发环境

要完成 YOLOv5 的 OpenVINO™推理程序开发，需要安装：YOLOv5 + openvino-dev。

由于 YOLOv5 的工程化做的实在太好，在 Windows10 中安装上述环境，只需要两步。

- 第一步：`git clone https://github.com/ultralytics/yolov5.git`
- 第二步：在 yolov5 文件夹中，修改 *requirements.txt* 文件的 25、30、34 和 35 行如下所示。

```
23  # Export ----------------------------------------
24  # coremltools>=4.1  # CoreML export
25  onnx  # ONNX export
26  # onnx-simplifier>=0.3.6  # ONNX simplifier
27  # scikit-learn==0.19.2  # CoreML quantization
28  # tensorflow>=2.4.1  # TFLite export
29  # tensorflowjs>=3.9.0  # TF.js export
30  openvino-dev  # OpenVINO export
31
32  # Extras ----------------------------------------
33  # albumentations>=1.0.3
34  Cython  # for pycocotools https://github.com/cocodataset/cocoapi/issues/172
35  pycocotools  # COCO mAP
36  # roboflow
37  thop  # FLOPs computation
```

然后使用命令：

```
pip install -r requirements.txt
```

完成开发环境安装，如图 1-6 所示。

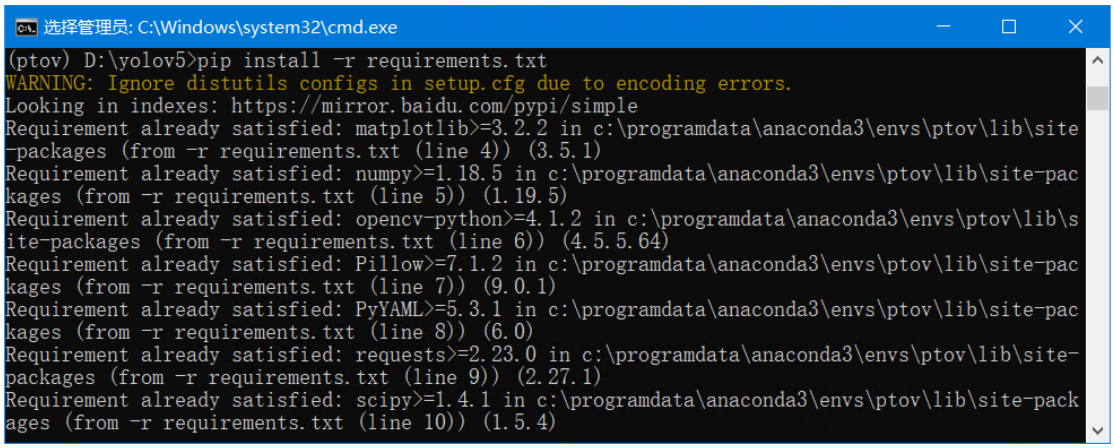


图 1-6　安装 YOLOv5 和 OpenVINO™ 开发环境

1.2.3 导出 YOLOv5 ONNX 模型

在 yolov5 文件夹下，使用命令

```
python export.py --weights yolov5s.pt --include onnx
```

完成 yolov5s.onnx 模型导出，如图 1-7 所示。

```
(ptov) D:\yolov5>python export.py --weights yolov5s.pt --include onnx
export: data=data\coco128.yaml, weights=['yolov5s.pt'], imgsz=[640, 640], batch_size=1,
device=cpu, half=False, inplace=False, train=False, optimize=False, int8=False, dynamic=
False, simplify=False, opset=12, verbose=False, workspace=4, nms=False, agnostic_nms=Fal
se, topk_per_class=100, topk_all=100, iou_thres=0.45, conf_thres=0.25, include=['onnx']
YOLOv5  v6.1-39-g4effd06 torch 1.11.0 CPU

Fusing layers...
Model Summary: 213 layers, 7225885 parameters, 0 gradients

PyTorch: starting from yolov5s.pt with output shape (1, 25200, 85) (14.1 MB)

ONNX: starting export with onnx 1.11.0...
ONNX: export success, saved as yolov5s.onnx (28.0 MB)

Export complete (2.83s)
Results saved to D:\yolov5
Detect:          python detect.py --weights yolov5s.onnx
PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5s.onnx')
Validate:        python val.py --weights yolov5s.onnx
Visualize:       https://netron.app

(ptov) D:\yolov5>
```

图 1-7　导出 YOLOv5s.onnx

1.2.4 用 Netron 工具查看 YOLOv5s.onnx 的输入和输出

使用 Netron，查看 YOLOv5s.onnx 模型的输入和输出，如图 1-8 所示。

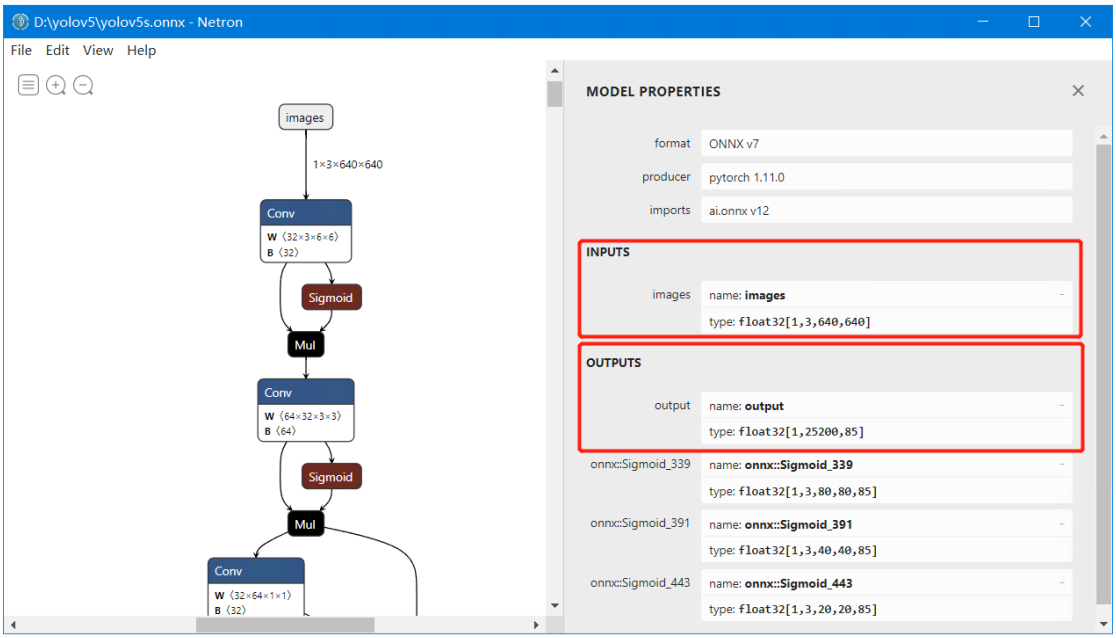


图 1-8　查看 YOLOv5s 模型输入输出

从图中可以看出：YOLOv5 6.1 版本之后，直接导出的 ONNX 格式模型会多出一个 output 层，这个 output 已经完全整合了之前三层的原始输出，再也不需要搞 anchor 的比率跟重新写解析后处理了，output 出来每一行 85 个数值，前面 5 个数值分别是：

cx, cy, w, h, score，后面 80 个参数是 MSCOCO 的分类得分

1.2.5 开发 YOLOv5 的 OpenVINO™推理程序

使用 OpenVINO™ 2022.1 开发推理程序的典型步骤如下所示，从图 1-9 中可见，三行 OpenVINO API 函数可以完成 YOLOv5s.onnx 模型的推理。

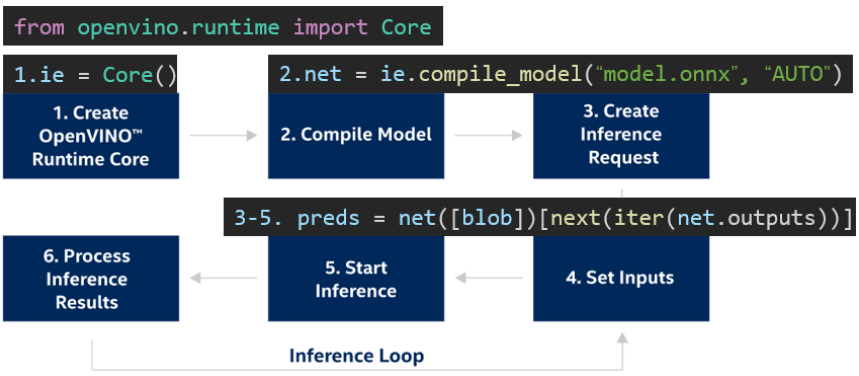


图 1-9　OpenVINO™ 推理程序开发步骤

完整范例程序如下：

```
# Do the inference by OpenVINO2022.1
from pyexpat import model
import cv2
import numpy as np
import time
import yaml
from openvino.runtime import Core  # the version of openvino >= 2022.1

# 载入 COCO Label
with open('./coco.yaml','r', encoding='utf-8') as f:
    result = yaml.load(f.read(),Loader=yaml.FullLoader)
class_list = result['names']

# YOLOv5s 输入尺寸
INPUT_WIDTH = 640
INPUT_HEIGHT = 640

# 目标检测函数，返回检测结果
def detect(image, net):
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (INPUT_WIDTH,
INPUT_HEIGHT), swapRB=True, crop=False)
    preds = net([blob])[next(iter(net.outputs))] # API version>=2022.1
    return preds

# YOLOv5 的后处理函数，解析模型的输出
def wrap_detection(input_image, output_data):
    class_ids = []
    confidences = []
    boxes = []
```

```
    #print(output_data.shape)
    rows = output_data.shape[0]

    image_width, image_height, _ = input_image.shape

    x_factor = image_width / INPUT_WIDTH
    y_factor = image_height / INPUT_HEIGHT

    for r in range(rows):
        row = output_data[r]
        confidence = row[4]
        if confidence >= 0.4:

            classes_scores = row[5:]
            _, _, _, max_indx = cv2.minMaxLoc(classes_scores)
            class_id = max_indx[1]
            if (classes_scores[class_id] > .25):
                confidences.append(confidence)

                class_ids.append(class_id)

                x, y, w, h = row[0].item(), row[1].item(),
row[2].item(), row[3].item()
                left = int((x - 0.5 * w) * x_factor)
                top = int((y - 0.5 * h) * y_factor)
                width = int(w * x_factor)
                height = int(h * y_factor)
                box = np.array([left, top, width, height])
                boxes.append(box)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.25, 0.45)

    result_class_ids = []
    result_confidences = []
    result_boxes = []

    for i in indexes:
        result_confidences.append(confidences[i])
        result_class_ids.append(class_ids[i])
        result_boxes.append(boxes[i])

    return result_class_ids, result_confidences, result_boxes

# 按照YOLOv5 letterbox Resize要求，先将图像长:宽 = 1:1，多余部分填充黑边
```

```
def format_yolov5(frame):
    row, col, _ = frame.shape
    _max = max(col, row)
    result = np.zeros((_max, _max, 3), np.uint8)
    result[0:row, 0:col] = frame
    return result

# 载入 yolov5s onnx 模型
model_path = "./yolov5s.onnx"
ie = Core() #Initialize Core version>=2022.1
net = ie.compile_model(model=model_path, device_name="AUTO")

# 开启 Webcam，并设置为 1280x720
cap = cv2.VideoCapture(0)
# 调色板
colors = [(255, 255, 0), (0, 255, 0), (0, 255, 255), (255, 0, 0)]

# 开启检测循环
while True:
    start = time.time()
    _, frame = cap.read()
    if frame is None:
        print("End of stream")
        break
    # 将图像按最大边 1:1 放缩
    inputImage = format_yolov5(frame)
    # 执行推理计算
    outs = detect(inputImage, net)
    # 拆解推理结果
    class_ids, confidences, boxes = wrap_detection(inputImage, outs[0])

    # 显示检测框 bbox
    for (classid, confidence, box) in zip(class_ids, confidences,
boxes):
        color = colors[int(classid) % len(colors)]
        cv2.rectangle(frame, box, color, 2)
        cv2.rectangle(frame, (box[0], box[1] - 20), (box[0] + box[2],
box[1]), color, -1)
        cv2.putText(frame, class_list[classid], (box[0], box[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 0, 0))

    # 显示推理速度 FPS
    end = time.time()
    inf_end = end - start
```

```
    fps = 1 / inf_end
    fps_label = "FPS: %.2f" % fps
    cv2.putText(frame, fps_label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)
    print(fps_label+ "; Detections: " + str(len(class_ids)))
    cv2.imshow("output", frame)

    if cv2.waitKey(1) > -1:
        print("finished by user")
        break
```

上述范例程序下载链接：https://gitee.com/ppov-nuc/yolov5_infer/blob/main/infer_by_openvino2022.py

运行效果如下所示：

```
(ptov) D:\yolov5\yolov5_infer_demo>python infer_by_OpenVINO2022.1.py
FPS: 1.16; Detections: 3
FPS: 6.54; Detections: 4
FPS: 7.07; Detections: 3
FPS: 7.09; Detections: 4
FPS: 7.04; Detections: 5
FPS: 6.69; Detections: 4
FPS: 7.12; Detections: 4
FPS: 6.26; Detections: 4
FPS: 6.94; Detections: 5
FPS: 7.43; Detections: 4
FPS: 6.91; Detections: 4
FPS: 7.47; Detections: 3
```

到此，完成了使用 OpenVINO™ 开发 YOLOv5 模型推理程。

1.3 运行带 ds-mqtt 的 EdgeX Foundry

首先，请安装好 docker 和 docker-compose。本文的运行环境是：Ubuntu20.04.4 LTS + Intel® i5-1135G7。

然后，请克隆 edgex-compose 代码仓到本地，使用命令：

```
git clone https://github.com/edgexfoundry/edgex-compose.git
```

运行结果，如图 1-10 所示。

```
(base) nuc@nuc-NUC11PAHi5:~/Downloads/edgex_ov$ sudo git clone https://gitee.com/ppov-nuc/edgex-compose.git
Cloning into 'edgex-compose'...
Username for 'https://gitee.com': 18675388132
Password for 'https://18675388132@gitee.com':
remote: Enumerating objects: 2046, done.
remote: Counting objects: 100% (2046/2046), done.
remote: Compressing objects: 100% (385/385), done.
remote: Total 2046 (delta 1650), reused 2046 (delta 1650), pack-reused 0
Receiving objects: 100% (2046/2046), 483.38 KiB | 418.00 KiB/s, done.
Resolving deltas: 100% (1650/1650), done.
(base) nuc@nuc-NUC11PAHi5:~/Downloads/edgex_ov$ ls
edgex-compose  Yolov5
(base) nuc@nuc-NUC11PAHi5:~/Downloads/edgex_ov$ cd edgex-compose/compose-builder/
(base) nuc@nuc-NUC11PAHi5:~/Downloads/edgex_ov/edgex-compose/compose-builder$
```

图 1-10　克隆 edgex-compose 代码仓

接着，在 compose-builder 目录下运行命令：

```
make gen no-secty ds-mqtt mqtt-broker
```

生成 edgex 的 docker compose 文件，如图 1-11 所示。

图 1-11　生成 docker compose 文件

最后，在 compose-builder 目录下运行命令：

```
docker-compose up -d
```

启动带有 MQTT Broker 的 EdgeX Foundry，如图 1-12 所示。

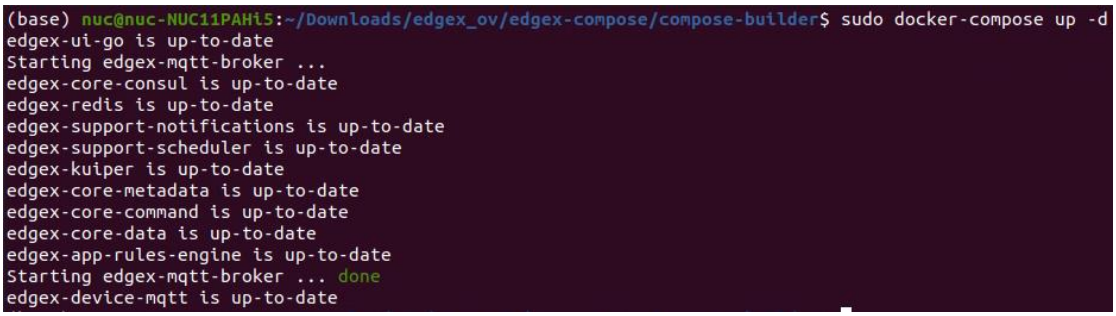


图 1-12 启动带有 MQTT Broker 的 EdgeX Foundry

到此，成功启动带有 MQTT Broker 的 EdgeX Foundry。

1.4　在 EdgeX UI 中新建 MQTT Device

为了简化 MQTT device profile 的编写过程，请先克隆张华乔老师的 mock-device-driver 代码仓到本地，使用命令：

```
git clone https://github.com/badboy-huaqiao/mock-device-driver.git
```

1.4.1 添加 device profile

第一步，启动 EdgeX UI。EdgeX UI 是一个图形化的 EdgeX 管理工具，在浏览器中，输入 *localhost:4000*; 然后在 Device Profile 中，点击“Add”按钮，如图 1-13 所示。

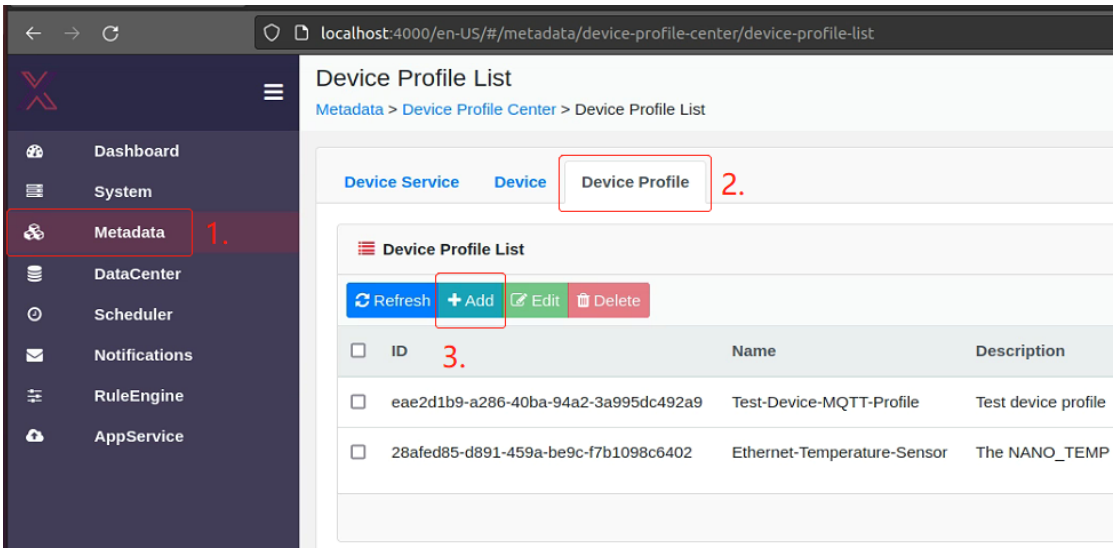


图 1-13　启动 EdgeX UI

第二步，在 mock-device-driver/jakarta-v2.1.0 文件夹下，有一个 test-mqtt-profile.yml 文件，请将该文件拖入“**Add Profile**”框，然后点“**Submit**”按钮，完成 mqtt device profile 的添加，如图 1-14 所示。

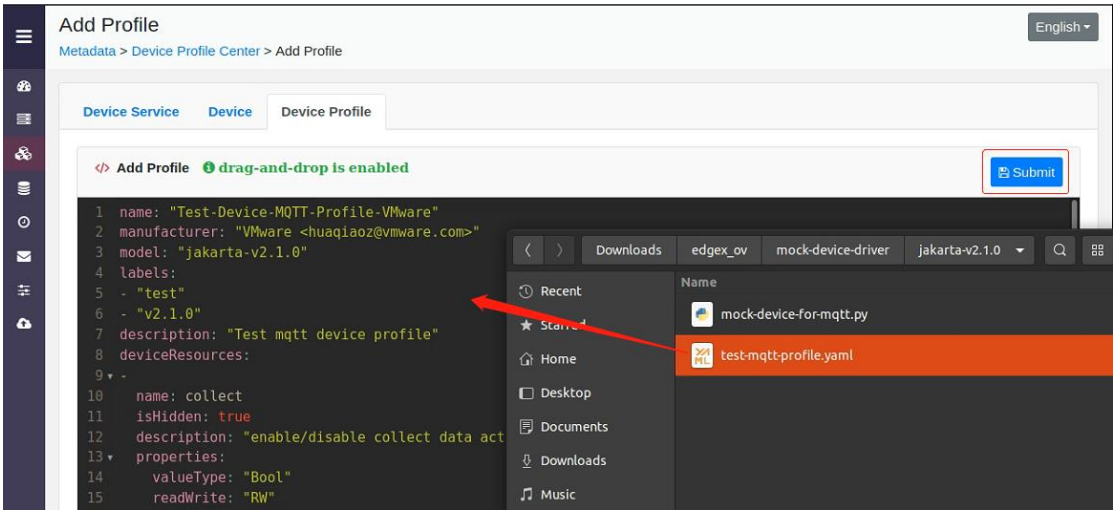


图 1-14　添加 test-mqtt-profile.yml 文件

1.4.2 新增一个名为“edgex_ov”的设备

第一步，在“Device”页面，点击“Add”按钮，如图 1-15 所示。

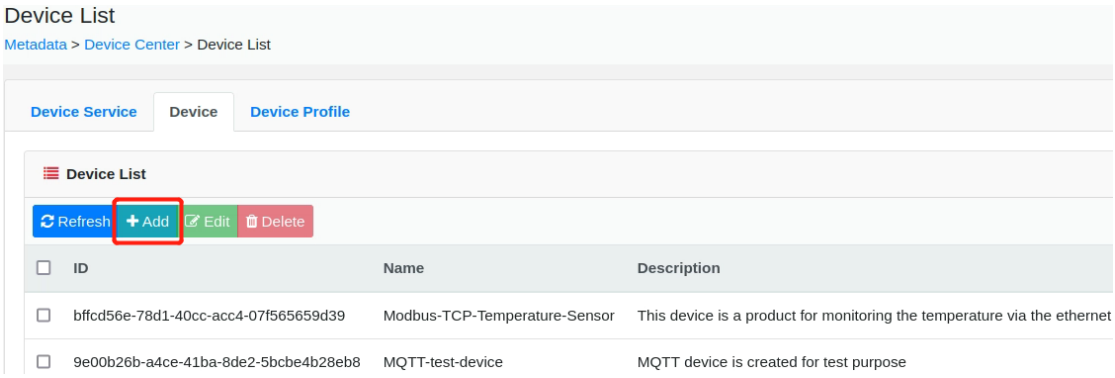


图 1-15　点击“Add”按钮

第二步，勾选“device-mqtt”，并按“Next”按钮，如图 1-16 所示。

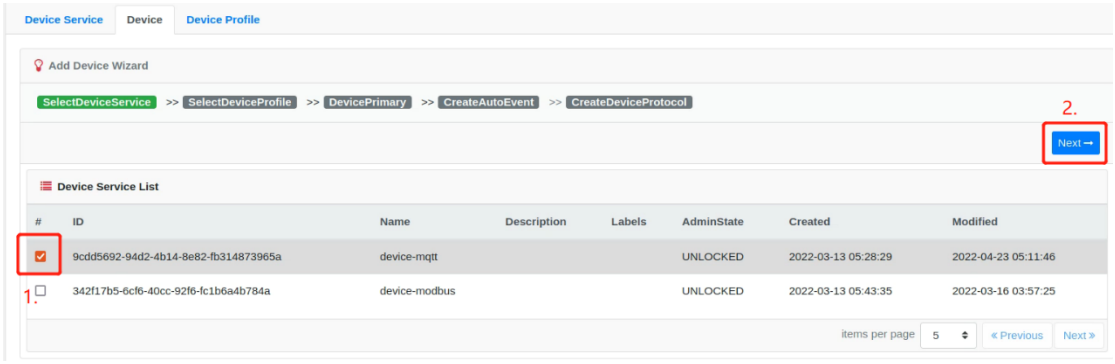


图 1-16　勾选“device-mqtt”

第三步，勾选“Test-Device-MQTT-Profile-VMware”，并按“Next”按钮，如图 1-17 所示。

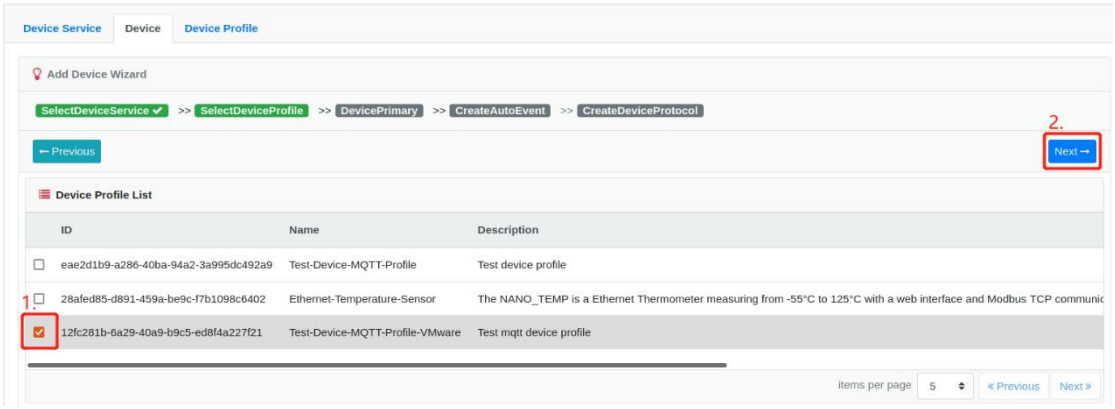


图 1-17　勾选“Test-Device-MQTT-Profile-VMware”

第四步，填写“Name”和“Description”，并按“Next”按钮，如所示。

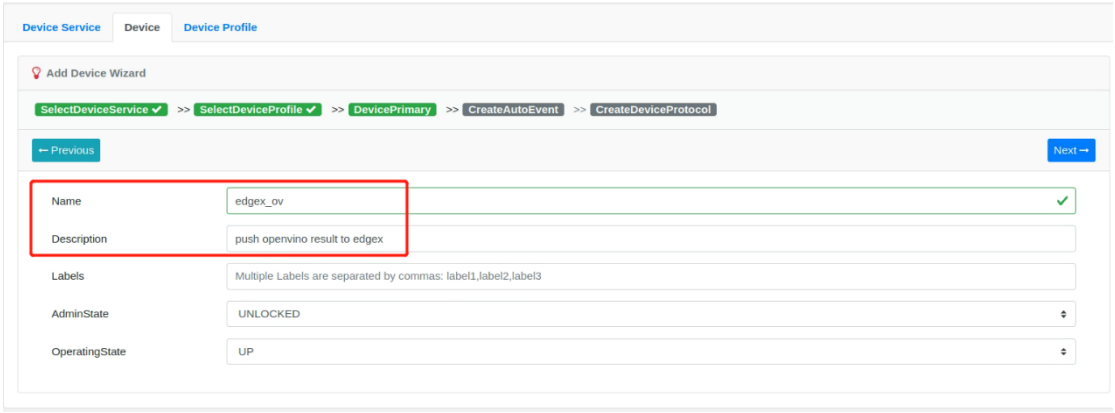


图 1-18　填写“Name”和“Description”

第五步，在“Add More AutoEvent”页面，按“Skip”按钮，跳过该设置。

第六步，在“CreateDeviceProtocol”页面，填写 Protocol 信息，然后按“Submit”按钮，完成名为“edgex_ov”的 MQTT 设备添加，如图 1-19 和图 1-20 所示。

- Protocol Name: device-mqtt
- Schema：tcp
- Host：0.0.0.0
- Port：1883
- User：huaqiaoz
- Password：1234
- ClientId: 123
- CommandTopic: CommandTopic

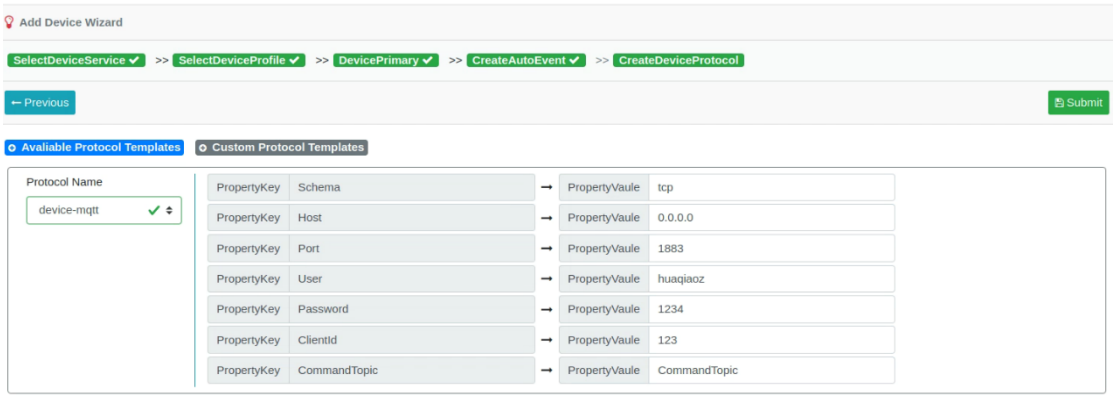


图 1-19　填写 Protocol 信息

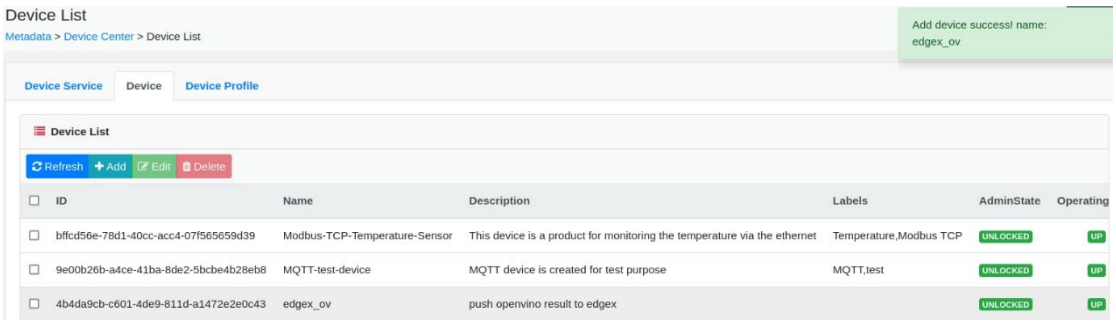


图 1-20　成功添加名为”edgex_ov”的 MQTT 设备

1.4.3 运行 mock-device-for-mqtt.py 测试添加的设备

添加了 MQTT 设备后，使用 mock-device-for-mqtt.py 的 Python 脚本来测试通讯是否正常，如图 1-21 所示。

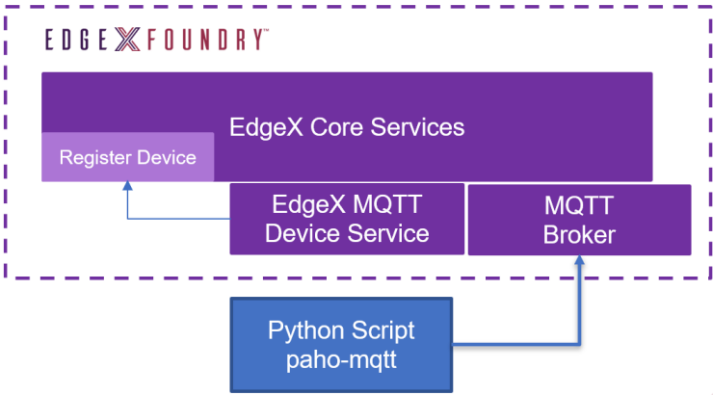


图 1-21　Python 脚本与 EdgeX 的通信构架

第一步：请先安装 paho-mqtt，参考：https://pypi.org/project/paho-mqtt/。

第二步：在 mock-device-driver/Jakarta-v2.1.0 文件夹下，运行命令，如图 1-22 所示。

```
python mock-device-for-mqtt.py
```

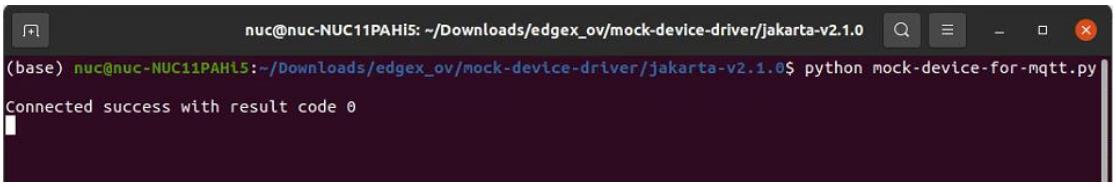


图 1-22　运行 mock-device-for-mqtt.py

第三步：在 EdgeX UI 中，“Device”页面，选择 testcollect，然后在“Set”部分，选择“true”，最后点击“try”按钮，可以使能设备，如图 1-23 所示。

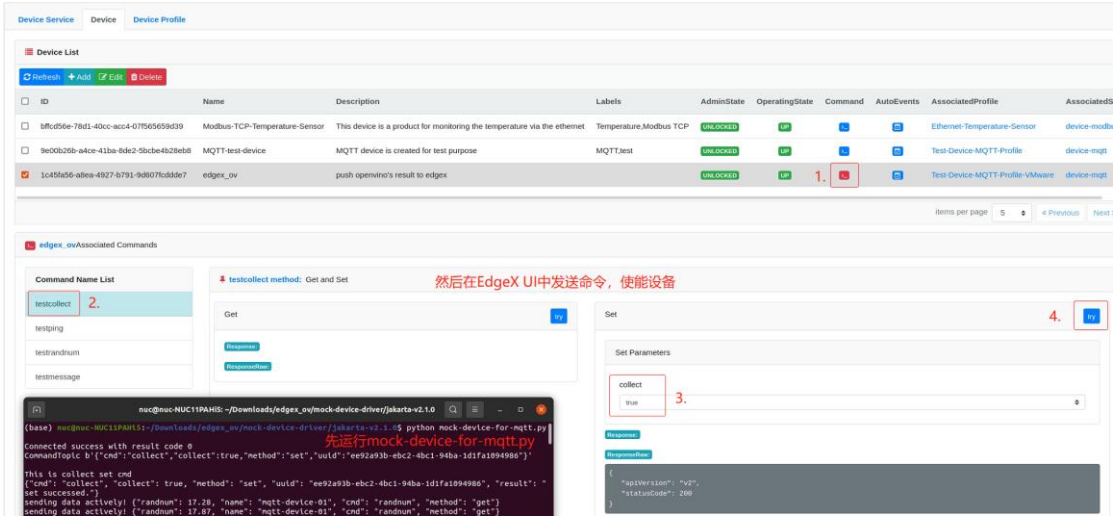


图 1-23　使能设备

第四步：运行 testmessage 命令，可以获得如图 1-24 所示结果。结果显示 EdgeX 获得了 Python 脚本发出的“Are you ok?”信息，新添加的“edgex_ov”MQTT 设备工作正常。

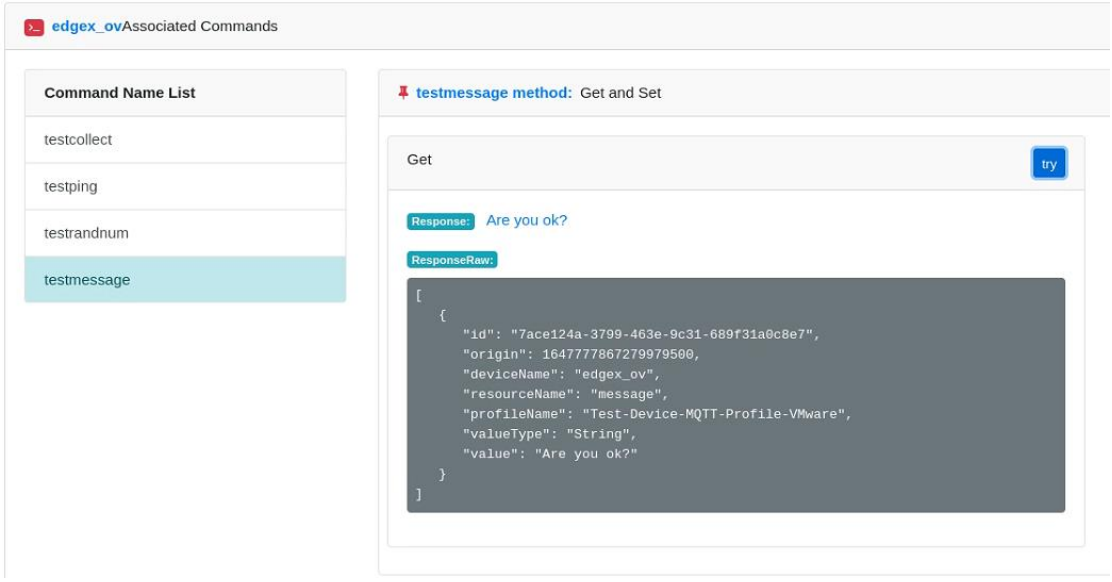


图 1-24　运行 testmessage 命令

1.5　将 OpenVINO™推理结果推送给 MQTT Broker

将 mock-device-for-mqtt.py 中的 on_message()函数，更新为 OpenVINO 的推理结果，如图 1-25 所示。

```
#当接收到命令，响应命令
#v2.1.0 get命令格式： {"cmd":"ping","method":"get","uuid":"f46ae5c7-2a08-4f56-a38b-d9e122d255d0"}
#v2.1.0 set命令格式 {"cmd":"ping","ping":"ping_set_value","method":"set","uuid":"95d559ad-7140-4d62-8dd8-6465c37fec10"}
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload)+'\n')
    d = json.loads(msg.payload)

    if d['cmd'] == "message":
        print("This is message cmd")
        if d['method'] == "get": # 执行OpenVINO推理程序，并拿到结果
            _, frame = cap.read()
            if frame is None:
                print("End of stream")
            # 将图像按最大边1:1放缩
            inputImage = format_yolov5(frame)
            # 执行推理计算
            outs = detect(inputImage, net)
            # 拆解推理结果
            class_ids, confidences, boxes = wrap_detection(inputImage, outs[0])
            msg = ""
            for (classid, confidence, box) in zip(class_ids, confidences, boxes):
                msg += f"detected: {class_list[classid]}:{confidence:.2f}, at xmin:{box[0]}, ymin:{box[1]}, xmax:{box[2]}, ymax:{box[3]}.\n"
            d['message'] = msg
            print(msg)
        elif d['method'] == "set":
            d['result'] = "set successed."
```

图 1-25　修改 on_message()函数

完整范例代码参见：

https://gitee.com/ppov-nuc/yolov5_infer/blob/main/openvino2022-device-for-mqtt.py

同样，参考 1.4.3 节，运行 openvino2022-device-for-mqtt.py，可以在 EdgeX UI 上，获得 OpenVINO 推理结果，如图 1-26 所示。

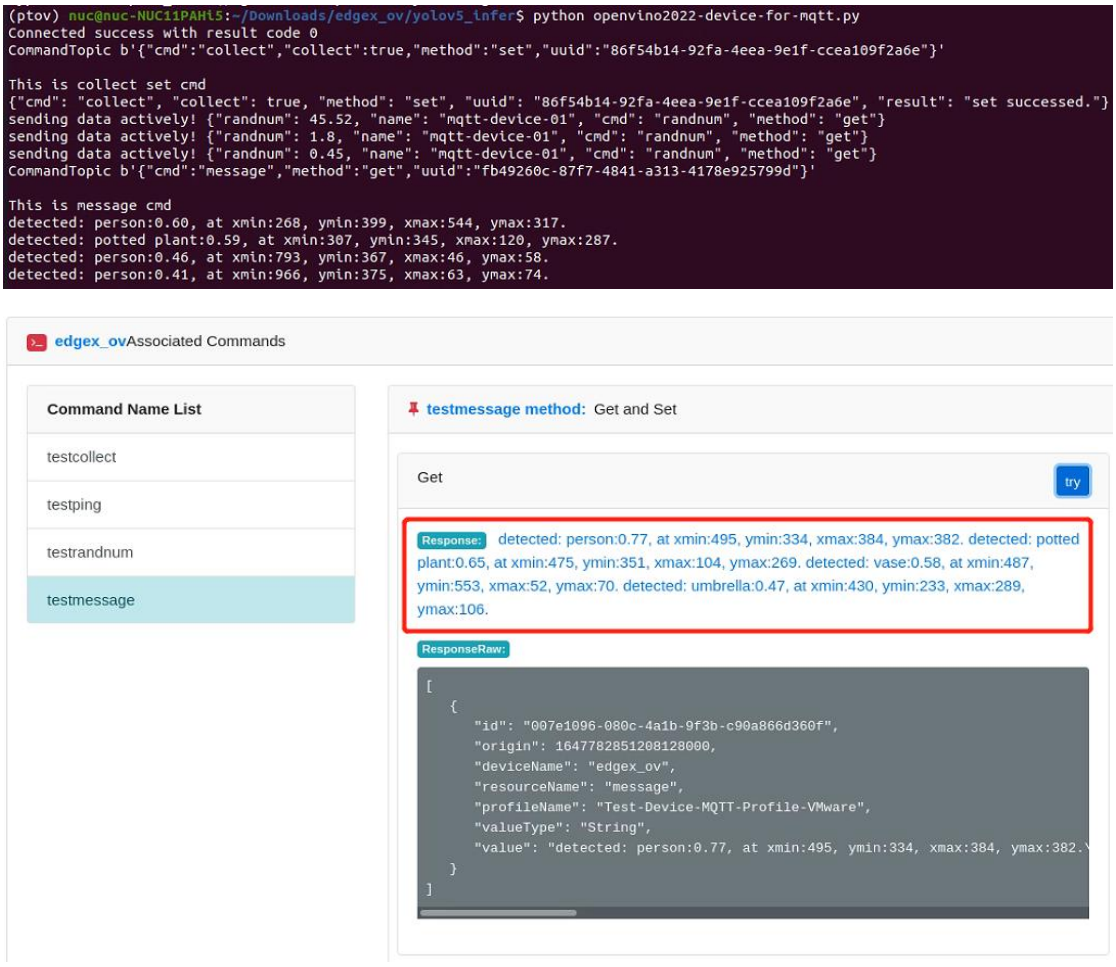


图 1-26　openvino2022-device-for-mqtt.py 运行结果

到此，将 OpenVINO™ 推理结果通过 MQTT 推送给 EdgeX Foundry 介绍完毕。

1.6　总结

基于 EdgeX Foundry 和 OpenVINO™ 工具包可以很方便的实现边缘智能，整个流程如图 1-27 所示。

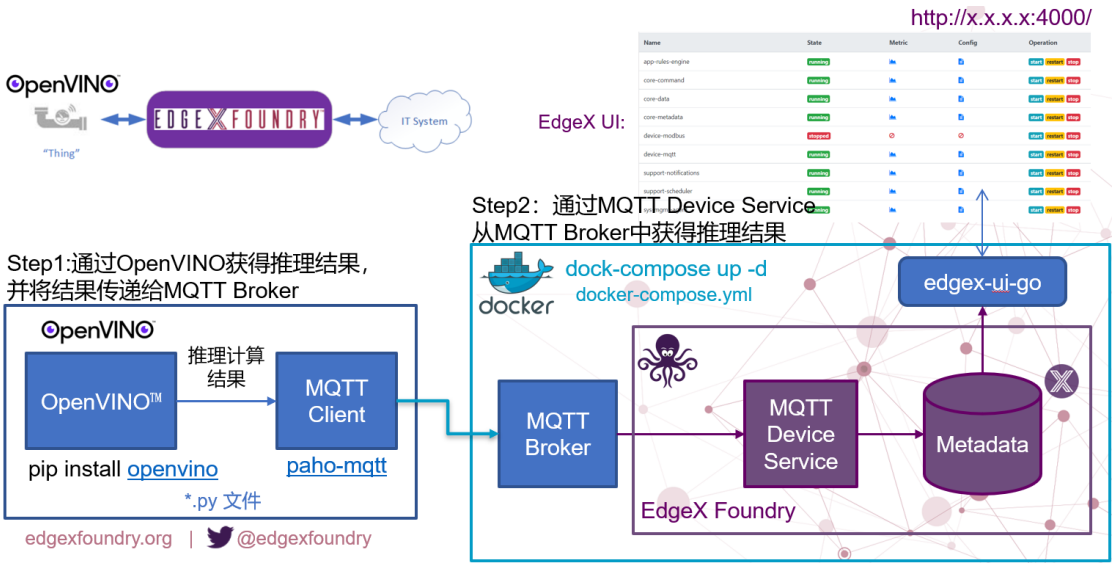


图 1-27　将 OpenVINO™ 推理结果通过 MQTT 推送给 EdgeX Foundry 的完整流程

参考资料：

- YOLOv5 模型： https://github.com/ultralytics/yolov5
- YOLOv5 推理程序：https://github.com/doleron/yolov5-opencv-cpp-python
- OpenVINO 推理范例：https://gitee.com/ppov-nuc/yolov5_infer
- Mock Device 范例：https://github.com/badboy-huaqiao/mock-device-driver
- OpenVINO APIs: https://docs.openvino.ai/latest/notebooks/002-openvino-api-with-output.html
- https://docs.openvino.ai/latest/openvino_docs_Integrate_OV_with_your_application.html#doxid-openvino-docs-integrate-o-v-with-your-application