

边缘系统设计与实践

- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

在边缘计算的模式中，服务器资源分布更加靠近请求源和数据源的网络边缘，使得物联网设备可以在其附近进行数据分析及决策。智慧家庭是物联网与边缘计算相结合的一个很好的应用场景。



由于摄像头在智慧家庭、安防等场景有着重要的应用，本节将图像处理任务作为一个特例，对所选择的前端设备及使用到的扩展硬件模块和服务器的环境配置及配置部署流程进行具体介绍。



- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

架构设计与硬件设备介绍 (1)

6

在整个结构中，分为前端物联网设备、通信技术、边缘服务器三个部分。

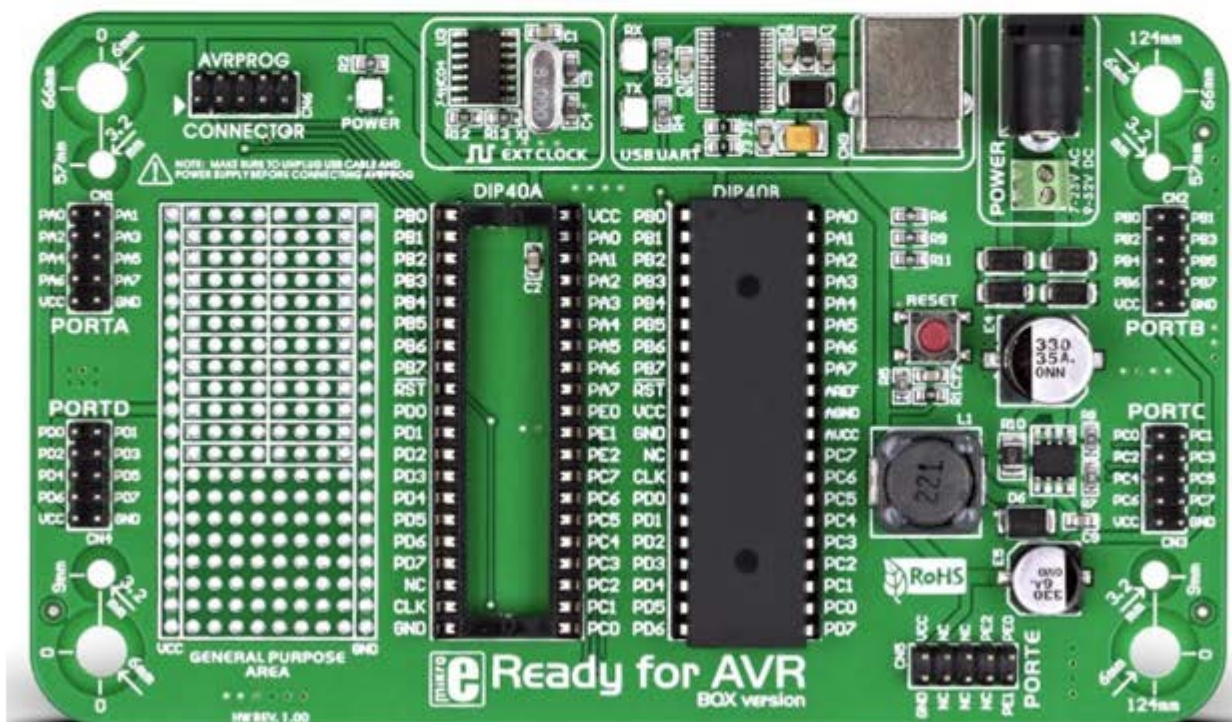


- ▶ 嵌入式系统或单片机的硬件尤其是其处理器很大程度上决定了该嵌入式设备能够完成的功能，在嵌入式或单片机中，常见的处理器有：
 - AVR
 - MSP430
 - ARM

架构设计与硬件设备介绍 (3)

8

AVR系列由Atmel公司于1996年研发，具有高性能、高速度、低功耗的特点。同时首次采用闪存作为数据存储介质的单芯片微控制器。



AVR家族

tinyAVR	megaAVR
megaAVR	Application-specific AVR
FPSLIC	32-bits AVR

MSP430系列是美国德州仪器(TI)1996年开始推向市场的一种16位超低功耗的微控制器，不同的MSP430系列集成了不同的外设，为数据采集以及数据分析提供了强大且灵活的平台。

MSP430x3xx	MSP430x1xx	MSP430F2xx	MSP430G2xx
MSP430x4xx	MSP430x5xx	MSP430C09x	MSP430L092

ARM处理器是基于ARM架构所生产的处理器的统称，基于RISC指令集，具有功耗低，体积小、成本低和性能高的特点，广泛应用于各类嵌入式设备中。

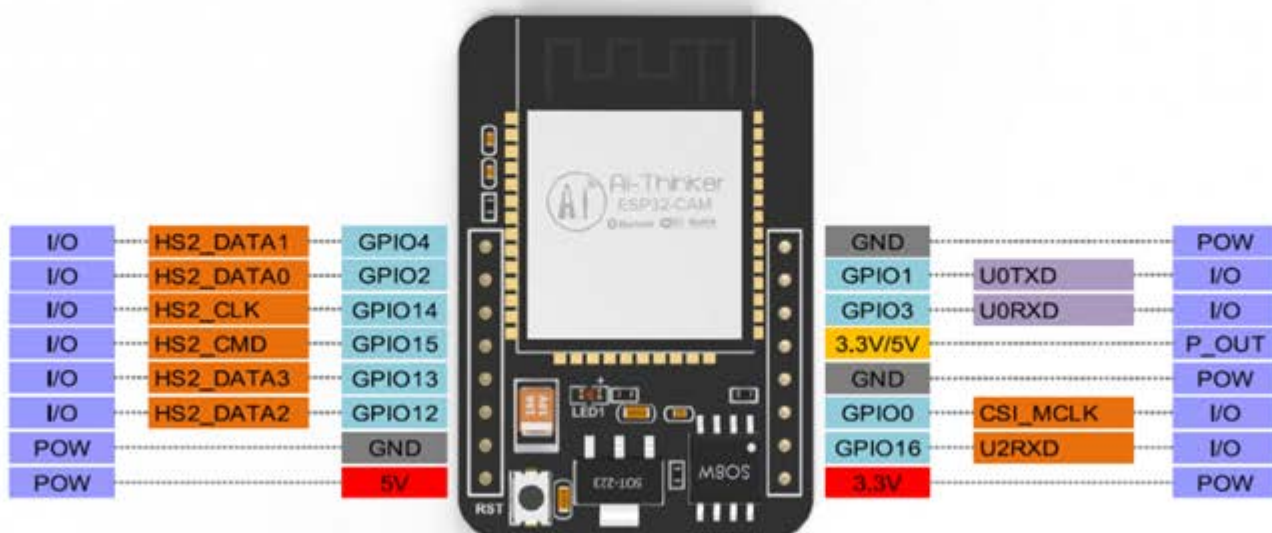
ARMv1	ARMv2	ARMv3	ARMv4
ARMv5	ARMv6	ARMv7	ARMv8
Cortex-A	Cortex-R	Cortex-M	

- ▶ **设备选择：**在这三类产品中，基于AVR的Arduino平台易于编程、对于初学者较为友好。Arduino与其他平台相比拥有如下优势。

基于C/C++，降低了开发难度
更便宜
跨平台
开源

- ▶ 然而由于Arduino开发板大多较为基础，且板载内存大小极为有限，在本章将采用集摄像头模块、无线模块为一体的Esp32-Cam作为前端设备。

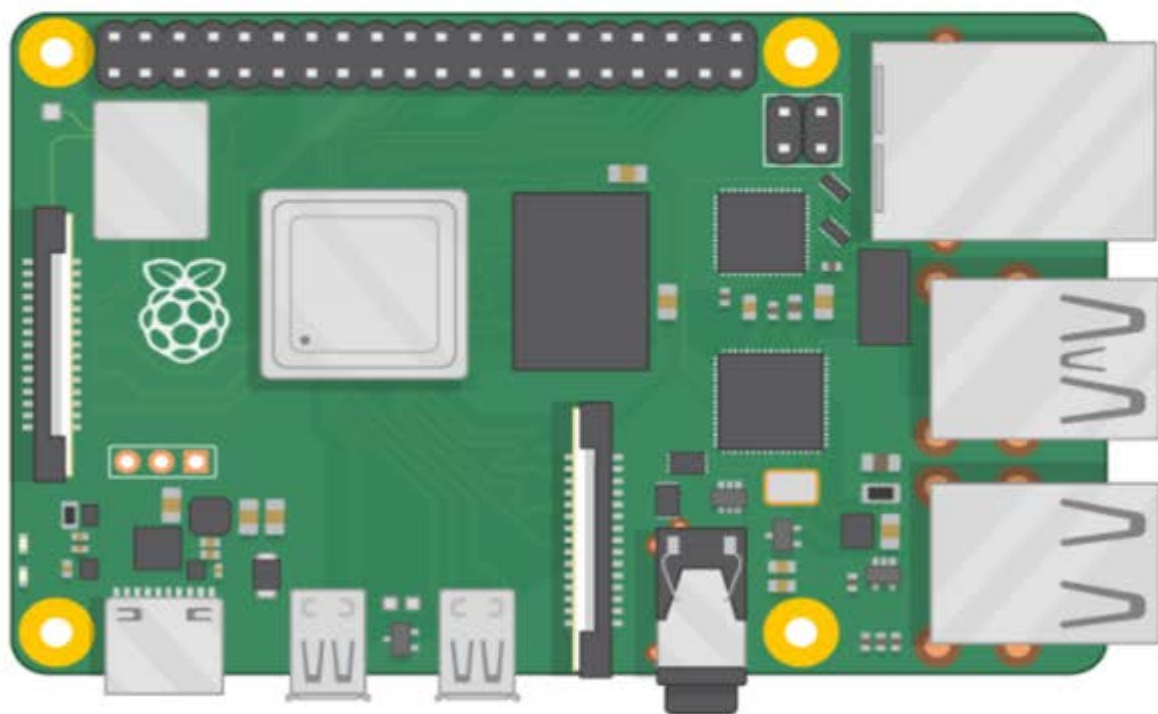
- ▶ Esp32-Cam是安信可基于乐鑫的Esp32芯片推出的一款小尺寸摄像头模组。



- ▶ Esp32-Cam拥有诸多优良特性，其内部集成了Wi-Fi模块和蓝牙模块，方便开发者直接使用。Esp32-Cam模块支持Station、AP、Station+AP三种工作模式。

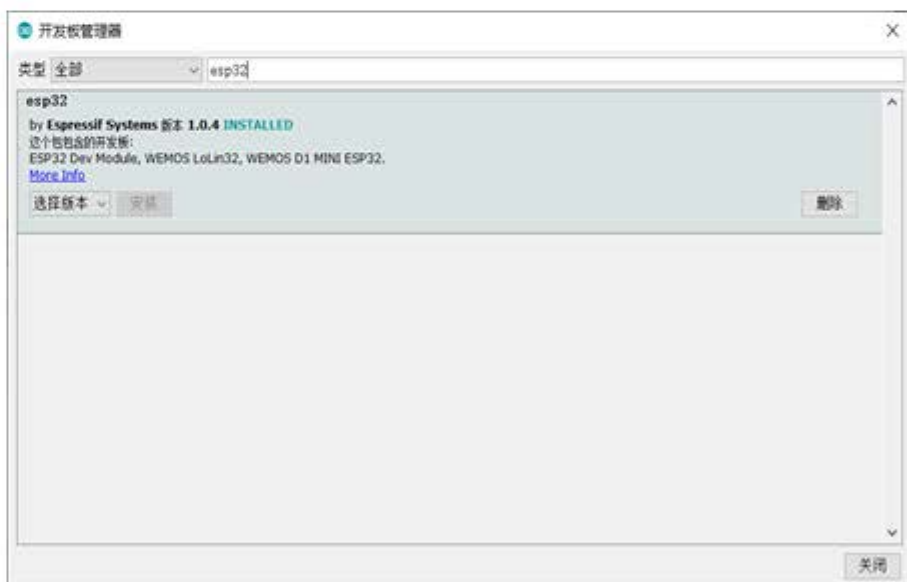


- ▶ 基于ARM平台的树莓派在可用性和易用性方面非常优秀，并且拥有体积小巧、功耗较低、性能强大、成本低等特点。

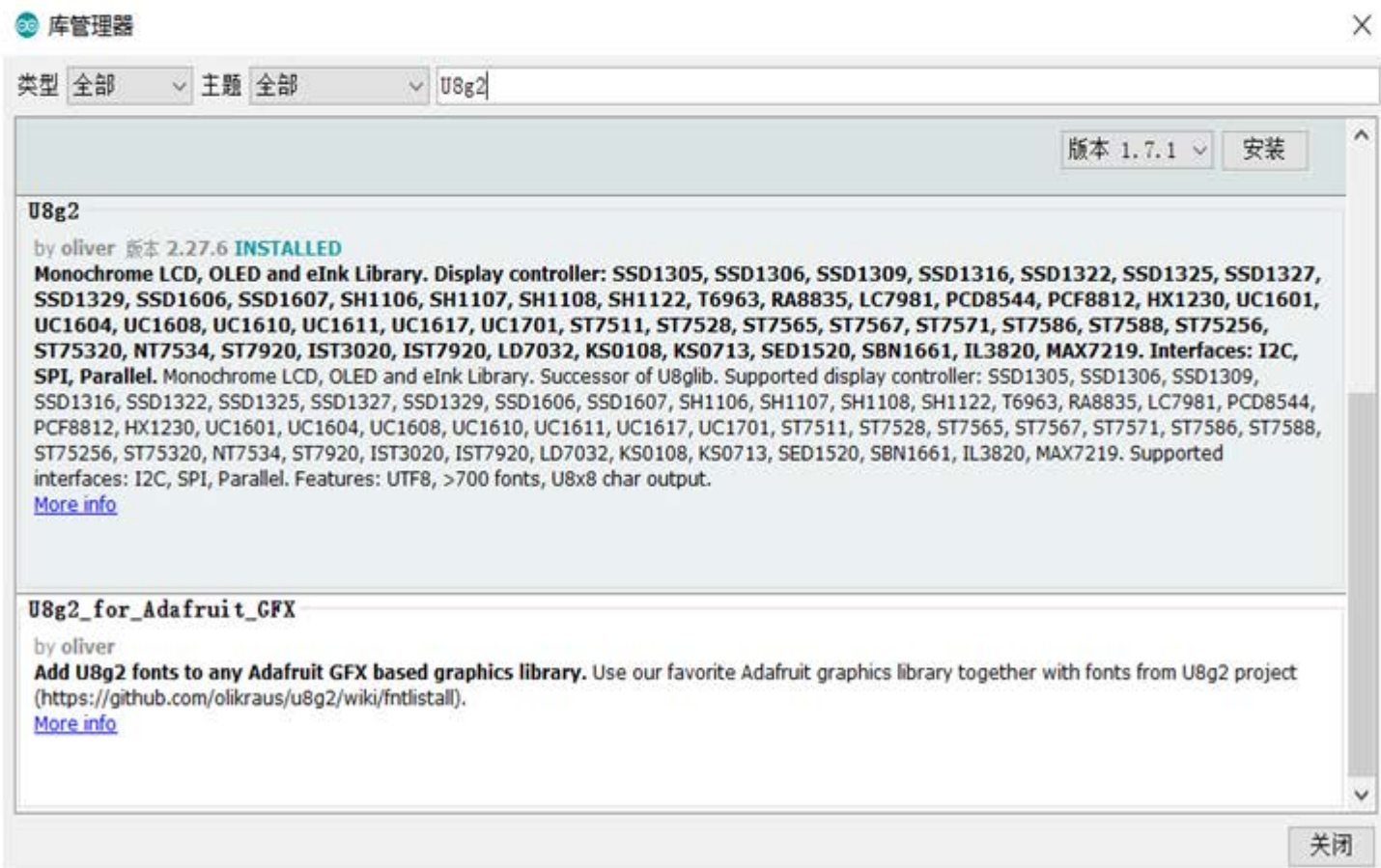


- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

本节将使用Arduino IDE作为开发工具，这是ARDUINO官方推出的开发软件，主要用于arduino程序的开发调试，并提供将编译好的代码直接烧录到开发板的功能。Arduino IDE没有自带Esp32-Cam的库函数，需要手动下载，下载步骤如下图所示。



接下来对SSD1306显示屏所需要的库进行配置。



下面介绍如何将代码烧录到Esp32-Cam开发板中，使用USB-TTL转换线将Esp32-Cam与电脑连接起来，具体的接线方式如下。

Esp32-Cam	USB-TTL
GND	GND
5V	5V
U0R	TX
U0T	RX
IO 0	GND

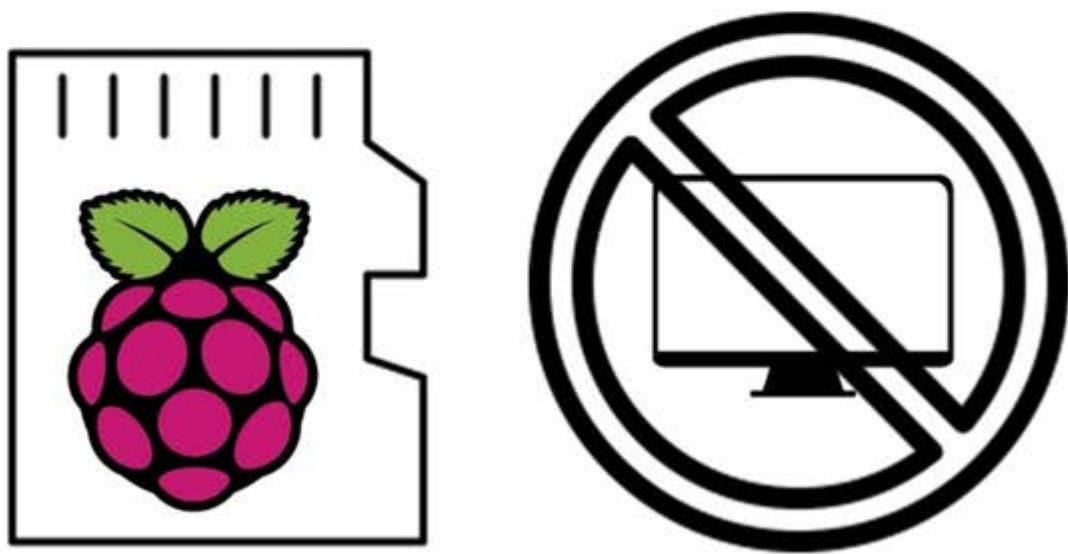
SSD1306显示屏与Esp32-Cam接线方式如下所示。

Esp32-Cam	SSD1306
GND	GND
3.3V	VCC
IO 14	SCL
IO 15	SDA

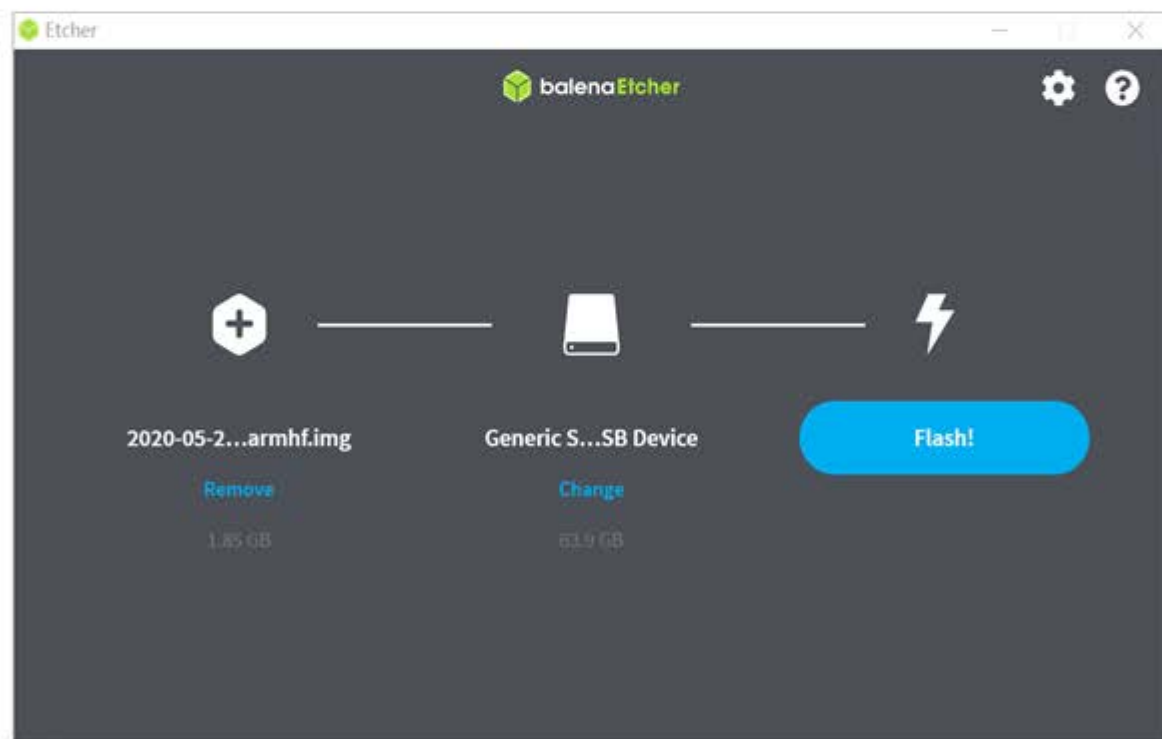
- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

- ▶ 系统安装
- ▶ Web服务器的搭建
- ▶ 视觉算法介绍
- ▶ TensorFlow Lite的安装及测试

- ▶ 树莓派官网列出了可选的系统镜像，包括Raspberry Pi OS(Raspbian)、Ubuntu Server、Ubuntu MATE、Ubuntu Core等。在本章中将使用官方推荐的Raspberry Pi OS作为基础系统。



- ▶ 首先准备一个USB读卡器和一张SD卡作为树莓派运行的磁盘介质，然后通过Windows或其他平台的电脑使用烧录软件（如：Etcher）将下载好的系统镜像烧录到SD卡中。



- ▶ 进入boot分区，在boot分区中新建两个文件。
- ▶ 最后插上电源，树莓派会自动开机并连接网络，此时可利用Xshell等软件连接到树莓派，系统默认用户名为：pi，密码为：raspberry。

- ▶ 系统安装
- ▶ Web服务器的搭建
- ▶ 视觉算法介绍
- ▶ TensorFlow Lite的安装及测试

- ▶ 在树莓派中采用python语言，利用Flask框架搭建简易的web server 用于监听前端设备的请求，并对前端设备所请求的服务进行处理。
- ▶ 在python web开发中，常见的开发框架有以下三种： Django、Tornado、和Flask。

- ▶ 下面介绍Flask框架的安装并进行简单测试。
 - 首先对树莓派系统进行换源操作。
 - 进行相应更新。
 - 安装pip3、安装Flask框架。
 - 打开浏览器访问 “http://ip:5000/”（其中的ip替换为树莓派的ip地址），网页显示 “Hello World”字样，则安装成功。

- ▶ 系统安装
- ▶ Web服务器的搭建
- ▶ 视觉算法介绍
- ▶ TensorFlow Lite的安装及测试

- ▶ Google公司看到了边缘计算的广阔落地前景，希望能够将深度学习模型方便快速的部署在边缘设备上，推出了专门面向移动设备、嵌入式设备和IOT设备的轻量级开源深度学习框架TensorFlow Lite。
- ▶ TensorFlow Lite官方提供的目标检测实例代码实现了以下的功能场景，从易于实现的角度考虑，我们将以该实例代码为基础进行修改，把原有的从摄像头采集图像改为通过网络从前端嵌入式设备接收图像。

- ▶ 系统安装
- ▶ Web服务器的搭建
- ▶ 视觉算法介绍
- ▶ TensorFlow Lite的安装及测试

- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

- ▶ Docker安装
- ▶ Docker Hub
- ▶ 个性化Docker镜像
- ▶ 整合

Docker有多种安装方式，这里介绍离线安装的方式。首先下载所需的安装包，包括containerd.io、docker-ce-cli和docker-ce，结果如图所示。

```
pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ wget https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/containerd.io_1.2.6-3_armhf.deb
--2021-01-20 03:20:06-- https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/containerd.io_1.2.6-3_armhf.deb
Resolving download.docker.com (download.docker.com)... 13.224.164.68, 13.224.164.41, 13.224.164.97, ...
Connecting to download.docker.com (download.docker.com)|13.224.164.68|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13025874 (12M) [application/x-debian-package]
Saving to: 'containerd.io_1.2.6-3_armhf.deb'

containerd.io_1.2.6-3_arm 100%[=====] 12.42M 3.62MB/s in 3.4s

2021-01-20 03:20:10 (3.62 MB/s) - 'containerd.io_1.2.6-3_armhf.deb' saved [13025874/13025874]

pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ wget https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/docker-ce-cli_19.03.6-3-0-raspbian-buster_armhf.deb
--2021-01-20 03:20:18-- https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/docker-ce-cli_19.03.6-3-0-raspbian-buster_armhf.deb
Resolving download.docker.com (download.docker.com)... 13.225.103.86, 13.225.103.10, 13.225.103.65, ...
Connecting to download.docker.com (download.docker.com)|13.225.103.86|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 29524662 (28M) [application/x-debian-package]
Saving to: 'docker-ce-cli_19.03.6-3-0-raspbian-buster_armhf.deb'

docker-ce-cli_19.03.6-3-0 100%[=====] 28.16M 5.81MB/s in 7.1s

2021-01-20 03:20:19 (3.97 MB/s) - 'docker-ce-cli_19.03.6-3-0-raspbian-buster_armhf.deb' saved [29524662/29524662]

pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ wget https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/docker-ce_19.03.6-3-0-raspbian-buster_armhf.deb
--2021-01-20 03:20:19-- https://download.docker.com/linux/raspbian/dists/buster/pool/stable/armhf/docker-ce_19.03.6-3-0-raspbian-buster_armhf.deb
Resolving download.docker.com (download.docker.com)... 13.224.164.59, 13.224.164.41, 13.224.164.97, ...
Connecting to download.docker.com (download.docker.com)|13.224.164.59|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 14966902 (14M) [application/x-debian-package]
Saving to: 'docker-ce_19.03.6-3-0-raspbian-buster_armhf.deb'

docker-ce_19.03.6-3-0-ras 100%[=====] 14.27M 3.67MB/s in 3.9s

2021-01-20 03:20:24 (3.67 MB/s) - 'docker-ce_19.03.6-3-0-raspbian-buster_armhf.deb' saved [14966902/14966902]
```

下载完成后，使用dpkg命令安装相应的安装包，结果如图。

```
pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ sudo dpkg -i containerd.io_1.2.6-3_armhf.deb
Selecting previously unselected package containerd.io.
(Reading database ... 42209 files and directories currently installed.)
Preparing to unpack containerd.io_1.2.6-3_armhf.deb ...
Unpacking containerd.io (1.2.6-3) ...
Setting up containerd.io (1.2.6-3) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.
pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ sudo dpkg -i docker-ce-cli_19.03.6~3-0~raspbian-buster_armhf.deb
Selecting previously unselected package docker-ce-cli.
(Reading database ... 42219 files and directories currently installed.)
Preparing to unpack docker-ce-cli_19.03.6~3-0~raspbian-buster_armhf.deb ...
Unpacking docker-ce-cli (5:19.03.6~3-0~raspbian-buster) ...
Setting up docker-ce-cli (5:19.03.6~3-0~raspbian-buster) ...
Processing triggers for man-db (2.8.5-2) ...
pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ sudo dpkg -i docker-ce_19.03.6~3-0~raspbian-buster_armhf.deb
Selecting previously unselected package docker-ce.
(Reading database ... 42424 files and directories currently installed.)
Preparing to unpack docker-ce_19.03.6~3-0~raspbian-buster_armhf.deb ...
Unpacking docker-ce (5:19.03.6~3-0~raspbian-buster) ...
Setting up docker-ce (5:19.03.6~3-0~raspbian-buster) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Processing triggers for systemd (241-7~deb10u4+rpil) ...
```


Docker安装完成后，启动docker并设置docker开机自启动。为了方便使用，将当前用户加入到docker用户组中。至此，docker已安装完成，可输入“docker version”进行验证，若控制台正常打印出docker版本信息，则安装成功。

```
pi@raspberrypi:~/examples/lite/examples/object_detection/raspberry_pi $ docker version
Client: Docker Engine - Community
Version: 19.03.6
API version: 1.40
Go version: go1.12.16
Git commit: 369ce74
Built: Thu Feb 13 01:37:07 2020
OS/Arch: linux/arm
Experimental: false

Server: Docker Engine - Community
Engine:
Version: 19.03.6
API version: 1.40 (minimum version 1.12)
Go version: go1.12.16
Git commit: 369ce74
Built: Thu Feb 13 01:31:06 2020
OS/Arch: linux/arm
Experimental: false
containerd:
Version: 1.2.6
GitCommit: 894b81a4b802e4eb2a91d1ce216b8817763c29fb
runc:
Version: 1.0.0-rc8
GitCommit: 425e105d5a03fabd737a126ad93d62a9eede87f
docker-init:
Version: 0.18.0
GitCommit: fec3683
```


- ▶ Docker安装
- ▶ Docker Hub
- ▶ 个性化Docker镜像
- ▶ 整合

Docker Hub是Docker提供的用于查找和共享容器镜像的服务，Docker Hub主要有以下几个功能特性：仓库、团队组织、官方镜像、第三方镜像、构建、Webhooks。



- ▶ Docker安装
- ▶ Docker Hub
- ▶ 个性化Docker镜像
- ▶ 整合

- ▶ **构建Docker镜像有两种常见的方式：**
 - 将修改后的容器打包为docker镜像
 - 通过dockerfile构建镜像。
- ▶ **下面将分别以这两种方式构建包含我们本章所需目标检测服务的Docker镜像。**

► 将修改后的容器打包为docker镜像

- 首先选择合适的基础镜像，进入树莓派字符界面，从Docker Hub拉取相应镜像，下载验证完成后，输入“docker images”命令可查看当前本地拥有的所有镜像。

```
pi@raspberrypi:~ $ docker pull python:3.7-slim-buster
3.7-slim-buster: Pulling from library/python
be493c6a5984: Pull complete
bfeb15d1de80: Pull complete
cd67726cded7: Pull complete
1206fcb7c398: Pull complete
23f07415d563: Pull complete
Digest: sha256:e66eadf34c2d15148566c75bbe2e6ac3fed6f243b17cdcefd3ae3fd12acabc75
Status: Downloaded newer image for python:3.7-slim-buster
docker.io/library/python:3.7-slim-buster
pi@raspberrypi:~ $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.7-slim-buster	59e8b25fa1bd	7 days ago	86.1MB

► 将修改后的容器打包为docker镜像

- 接下来基于该镜像启动并进入docker容器，执行结果如图。

```
pi@raspberrypi:~ $ docker run -it -p 5002:5002 python:3.7-slim-buster /bin/bash
root@910c383db962:/#
```

- 此时已经进入docker容器，所有的操作也都会在docker容器中生效，而与树莓派本身的系统没有关联。执行相应的更新操作，并下载编译程序所必须的一些头文件及软件包，以便后续安装配置流程的进行。

► 将修改后的容器打包为docker镜像

- 使用pip安装TensorFlow Lite, 如图所示。

```
root@910c383db962:/# cd /root
root@910c383db962:~# pip install https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_armv7l.whl
Collecting tflite-runtime==2.1.0.post1
  Downloading https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_armv7l.whl (1.8 MB)
    |████████████████████| 1.8 MB 622 kB/s
Collecting numpy>=1.12.1
  Downloading numpy-1.19.5.zip (7.3 MB)
    |████████████████████| 7.3 MB 5.2 MB/s
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing wheel metadata ... done
Building wheels for collected packages: numpy
  Building wheel for numpy (PEP 517) ... done
  Created wheel for numpy: filename=numpy-1.19.5-cp37-cp37m-linux_armv7l.whl size=3903437 sha256=01147b3c7c4a22f9a372a29ffac66f7221c20e17b7730958cc1872d5426530d0
  Stored in directory: /root/.cache/pip/wheels/62/f7/b2/69a26275770f55d1e8d7355aaa221caf38536f5f0795390b8f
Successfully built numpy
Installing collected packages: numpy, tflite-runtime
Successfully installed numpy-1.19.5 tflite-runtime-2.1.0.post1
```

► 将修改后的容器打包为docker镜像

- 下载TensorFlow官方提供的实例代码。
- 修改requirements.txt文件中所包含的pip安装列表。
- 对该文件内容进行如下修改，然后退出并保存。

```
# Python packages required for classify_picamera.py  
numpy  
Flask  
#picamera  
Pillow
```

► 将修改后的容器打包为docker镜像

- 下载必要的包以及预训练好的目标检测模型文件。
- 下载完成后，此时将附件中的“附件/local/model/labels.txt”复制到model文件夹中，将附件中“附件/local/”文件夹下的detect_picamera.py、dog.jpg、main.py三个文件复制到当前文件夹。
- 在宿主机使用docker cp命令复制完代码后，执行detect_picamera.py文件，若出现正常的检测结果信息，则环境搭建正确。

► 将修改后的容器打包为docker镜像

- 测试简易Web服务器的main.py是否正常工作。
- 测试成功后，将核心代码移到/root/目录下，方便后续使用。
- 在容器中执行“exit”命令退出容器，在宿主机利用“docker commit”命令将该容器打包为docker镜像。
- 打包完成后，使用“docker images”即可看到相应镜像。

```
pi@raspberrypi:~ $ docker commit kind_chaum detect:v1
sha256:7e07b8112112e3b18859f2a9228018554d4bf720547c97e541afca21da9bac99
pi@raspberrypi:~ $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
detect	v1	7e07b8112112	35 seconds ago	457MB
python	3.7-slim-buster	59e8b25fa1bd	7 days ago	86.1MB

► 通过Dockerfile创建镜像

- Dockerfile是用于构建Docker镜像的常见方式，它是一个文本文件，里面包含了构造镜像所需的自定义指令和格式。
- DockerFile语法命令是跨平台统一的，本节将对Dockerfile中的一些常用语法进行简单介绍。

► 通过Dockerfile创建镜像

■ From指令格式

```
FROM [--platform=<platform>] <image> [AS <name>]  
或者FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]  
或者FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

■ RUN指令格式

shell格式: RUN <command>

exec格式: RUN ["executable", "param1", "param2"]

► 通过Dockerfile创建镜像

■ CMD指令格式

exec格式: `CMD ["executable","param1","param2"]`
ENTRYPOINT参数格式: `CMD ["param1","param2"]`
shell格式: `CMD command param1 param2`

■ ENTRYPOINT指令格式

exec格式: `ENTRYPOINT ["executable", "param1", "param2"]`
shell格式: `ENTRYPOINT command param1 param2`

► 通过Dockerfile创建镜像

■ EXPOSE指令格式

```
EXPOSE <port> [<port>/<protocol>...]
```

■ ENV指令格式

格式一：ENV <key> <value>

格式二：ENV <key1>=<value1> <key2>=<value2>...

► 通过Dockerfile创建镜像

■ COPY指令格式

格式一: COPY [--chown=<user>:<group>] <src>... <dest>

格式二: COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]

■ ADD指令格式

格式一: ADD [--chown=<user>:<group>] <src>... <dest>

格式二: ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]

► 通过Dockerfile创建镜像

- 将附件下载到树莓派的“/home/pi”目录下后，在“附件/dockerfile”文件夹下执行“docker build”命令，进行目标镜像的构建。镜像构建完成后，使用“docker images”命令查看镜像。

```
pi@raspberrypi:~ $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
detect	v2	f2a882540cef	9 minutes ago	402MB
detect	v1	7e07b8112112	39 minutes ago	457MB
python	3.7-slim-buster	59e8b25fa1bd	7 days ago	86.1MB

- ▶ Docker安装
- ▶ Docker Hub
- ▶ 个性化Docker镜像
- ▶ 整合

为了解决树莓派这样的边缘服务器的资源有限的问题，我们在树莓派宿主主机上额外运行了一个Web服务器，用于接收前端设备的所有请求，接收到前端设备的请求后，会从请求中解析出前端设备需要获取的服务类型，接着判断搭载该服务的Docker容器是否处于运行状态。

- ▶ 课程简介
- ▶ 架构设计与硬件设备介绍
- ▶ 前端硬件设备环境说明
- ▶ 服务器环境配置
- ▶ 服务部署
- ▶ 利用EdgeX Foundry开源平台创建服务

- ▶ EdgeX Fonudry介绍
- ▶ 环境配置

- ▶ **EdgeX Foundry 是一个位于网络边缘的开源，供应商中立，灵活和可互操作的软件平台，可与其他设备、传感器、执行器和其他IoT对象进行交互。**
- ▶ **EdgeX Foundry是开源微服务的集合，包括四个服务层和两个基础增强系统服务。**
- ▶ **EdgeX的主要工作从传感器和设备收集数据，并将其提供给北侧的应用程序和系统。**

- ▶ EdgeX Fonudry介绍
- ▶ 环境配置

► 环境配置分为以下几个步骤：

- 安装docker和docker-compose
- 安装EdgeX Foundry
- 启动EdgeX Foundry
- 创建设备
- 发送数据到EdgeX Foundry
- 通过MQTT导出数据
- 通过树莓派采集数据发送至EdgeX

► 安装docker和docker-compose

- 进入虚拟机, 更新系统
- 安装Docker-CE
- 安装docke-compose

► 安装EdgeX Foundry

- 为EdgeX Foundry的yml文件创建目录
- 使用下载docker-compose.yml文件
- 拉取并查看服务镜像

```
root@ubuntu:/home/geneva# docker images
REPOSITORY                                TAG                                IMAGE ID
colorchanger                              latest                            6ee0b24513a9
emqx/kuiper                               0.4.2-alpine                     07beedcd55a4
edgexfoundry/docker-device-rest-go        1.1.1                            5d8734bd1a55
edgexfoundry/docker-support-scheduler-go  1.2.1                            8ff56ef98c94
edgexfoundry/docker-sys-mgmt-agent-go     1.2.1                            8069d55f1860
edgexfoundry/docker-support-notifications-go 1.2.1                            81bfc8eb1799
edgexfoundry/docker-core-command-go       1.2.1                            c9e88b3bbd8b
edgexfoundry/docker-core-metadata-go      1.2.1                            a33bfd480526
edgexfoundry/docker-core-data-go          1.2.1                            f50adbca9cb2
edgexfoundry/docker-app-service-configurable 1.2.0                            5b8a3e288cc1
edgexfoundry/docker-app-service-configurable 1.1.0                            5d0eb56b32fa
edgexfoundry/docker-edgex-consul          1.2.0                            9af65397ea6d
redis                                      5.0.8-alpine                     5c5637d8a823
```

► 启动EdgeX Foundry

- 通过docker-compose启动EdgeX Foundry并查看当前容器
- 停止EdgeX Foundry

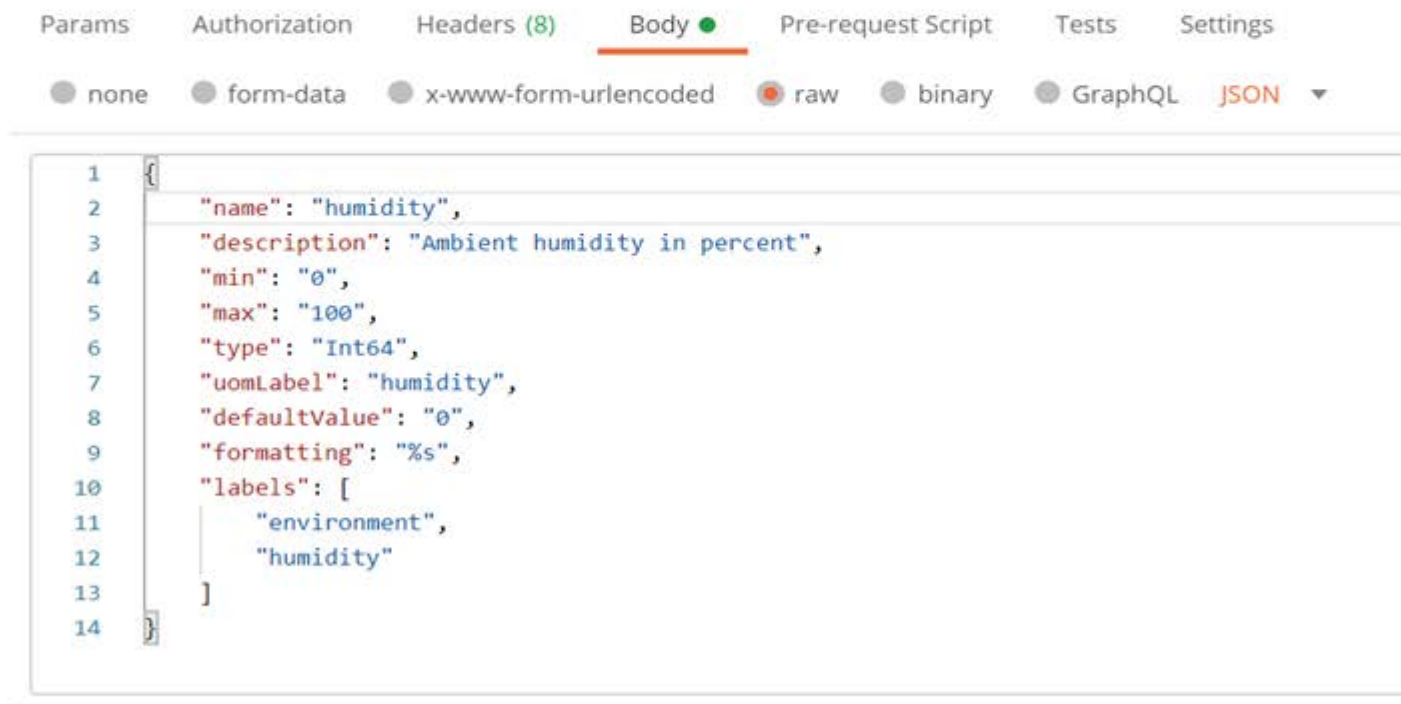
```
root@ubuntu:/home/geneva# docker-compose ps
```

Name	Command	State	Ports
edgex-app-service-configurable-mqtt	/app-service-configurable ...	Up	48095/tcp, 0.0.0.0:48101->48101/tcp
edgex-app-service-configurable-rules	/app-service-configurable ...	Up	48095/tcp, 0.0.0.0:48100->48100/tcp
edgex-core-command	/core-command -cp=consul.h ...	Up	0.0.0.0:48082->48082/tcp
edgex-core-consul	edgex-consul-entrypoint.sh ...	Up	8300/tcp, 8301/tcp, 8301/udp, 0.0.0.0:8400->8400/tcp, 0.0.0.0:8600->8600/udp
edgex-core-data	/core-data -cp=consul.http ...	Up	0.0.0.0:48080->48080/tcp, 0.0.0.0:48081->48081/tcp
edgex-core-metadata	/core-metadata -cp=consul. ...	Up	0.0.0.0:48081->48081/tcp
edgex-device-rest	/device-rest-go --cp=consu ...	Up	0.0.0.0:49986->49986/tcp
edgex-kuiper	/usr/bin/docker-entrypoint ...	Up	0.0.0.0:20498->20498/tcp, 0.0.0.0:20499->20499/tcp
edgex-redis	docker-entrypoint.sh redis ...	Up	0.0.0.0:6379->6379/tcp
edgex-support-notifications	/support-notifications -cp ...	Up	0.0.0.0:48060->48060/tcp
edgex-support-scheduler	/support-scheduler -cp=con ...	Up	0.0.0.0:48085->48085/tcp
edgex-sys-mgmt-agent	/sys-mgmt-agent -cp=consul ...	Up	0.0.0.0:48090->48090/tcp

```
root@ubuntu:/home/geneva#
```

► 创建设备

■ 创建value descriptors



► 创建设备

■ 上传device profile

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

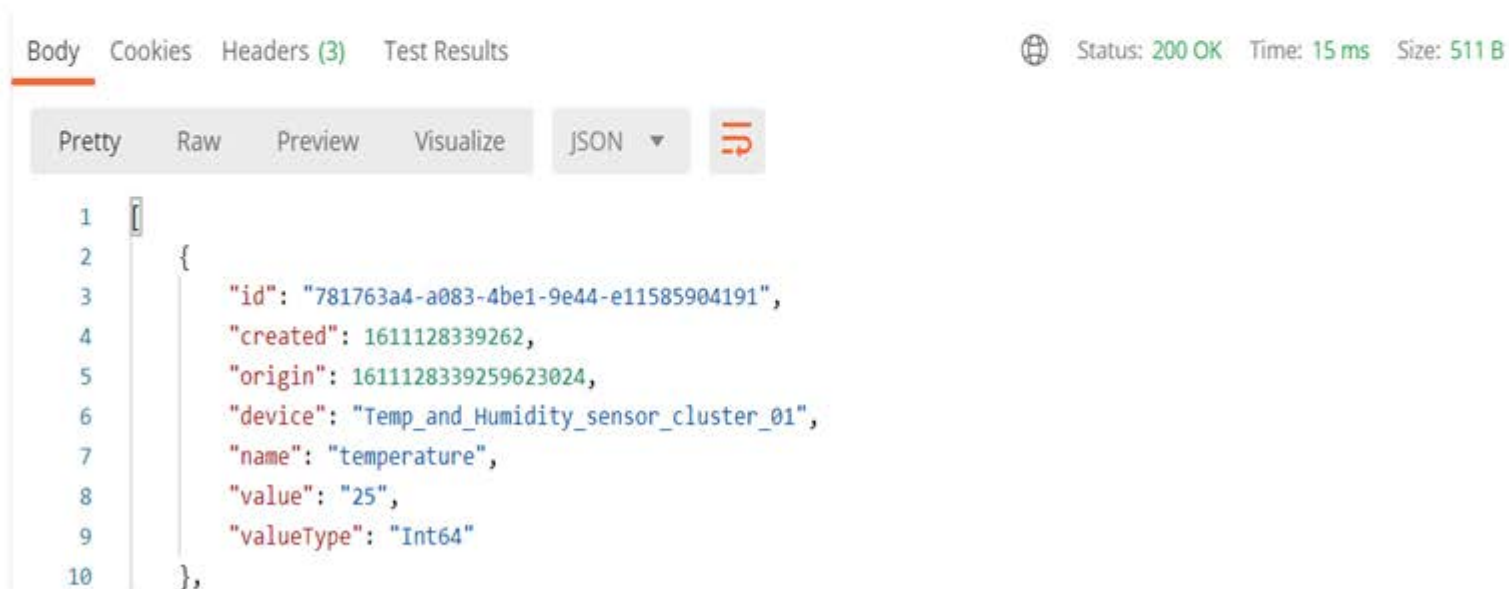
☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE
≡ <input checked="" type="checkbox"/>	file	File ▼ sensorClusterDeviceProfile.yml X
	Key	Value

■ 创建device

► 发送数据到EdgeX Foundry

- 现在EdgeX已经准备好接受温度和湿度数据了。首先我们先通过Postman尝试发送单独的数据。



Body Cookies Headers (3) Test Results

Status: 200 OK Time: 15 ms Size: 511 B

Pretty Raw Preview Visualize JSON

```
1  [
2    {
3      "id": "781763a4-a083-4be1-9e44-e11585904191",
4      "created": 1611128339262,
5      "origin": 1611128339259623024,
6      "device": "Temp_and_Humidity_sensor_cluster_01",
7      "name": "temperature",
8      "value": "25",
9      "valueType": "Int64"
10   },
```


► 通过MQTT导出数据

- 停止当前EdgeX
- 下载新的yaml配置文件
- 替换yaml配置文件
- 修改topic ID
- 启动EdgeX Foundry
- 打开HIVEMQ网页

- ▶ **通过树莓派采集数据发送至EdgeX**
 - 硬件选择与接线
 - 树莓派下载requirements文件和python代码
 - 安装并创建虚拟环境
 - 修改代码
 - 运行代码