**1). IoT Network Testbed Design**

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Devices** | **Attack Simulation** | **Tools** |
| IoT Device | |  | | --- | | Raspberry Pi (smart grid control aggregator node) | | Raspberry Pi Pico W (Simulated smart meter/filed sensor node) | |  | | DHT11 (temperature/humidity sensing) | | MQ2 (gas leak simulation) | | Smart Bulb (lighting control spoof) | | ESP32-CAM (video surveillance) | | Smartwatch (motion/fatigue feedback) | | Oura Ring / Biobeat (stress, heart rate, sleep detection) | | RFID Tags (access control simulation) | | Alexa (voice-command interface) | | |  | | --- | | sensor flooding (e.g., fake gas/temp values) | | Low & slow flooding | | Humidity spike injection | | Fake gas readings | | Blinking attack | | Frame spoofing / lag | | Fake high heart rate | | Fake stress, heart signal (in high or low rate)  Fake access logs triggering | | Repeated ambiguous voice commands | | Python script,  Micro-python script,  **Protocol:**  Application: HTTP/ MQTT/ CoAP  Transport: TCP/UDP  Security: TLS/DTLS |
| Edge Device | Raspberry pi  NVIDIA jetson nano | **Model Deployment:** Onnx runtime environment  **Local Server:** HTTP, MQTT, TCP, UDP  **Tools for traffic Capture:** Wireshark, TCP dump, Zeek, Suricata |  |
| Cloud/ Central server | Amazon AWS  High configuration computer |  |  |

**Attacking Tool**

1. **Python Script** (Scripting Language)  
   Used for simulating sensor behavior, flooding, spoofing, or slow-rate attacks.
2. **hping3** (Network Attack Tool)  
   A packet generator for TCP/UDP/ICMP floods; useful for DoS attacks or port scanning.
3. **THC-SSL-DOS** (DoS Attack Tool)  
   Simulates SSL-based DoS attacks by exhausting SSL handshakes on HTTPS endpoints or secured MQTT brokers.
4. **Node-RED** (Visual Automation Tool)  
   Used to create flows that emulate sensors, automate MQTT/HTTP flooding, and simulate device behaviors.
5. **MQTT.fx** (MQTT Client Tool)  
   GUI-based tool to manually publish/subscribe to MQTT topics and simulate IoT sensor data injection or flooding.
6. **Sandboxed** (Simulated Cloud Service / API Sandbox)  
   Used via Amazon Developer Console to simulate voice-command interactions and ambiguous command replay.
7. **Postman** (API Testing & Simulation Tool)  
   Used to send repeated or malformed HTTP requests to simulate attacks from smartwatches, Alexa, etc.
8. **Scapy** (Packet Crafting Tool / Python Library)  
   Used to craft and inject custom packets at network and transport layers (e.g., fake MQTT, Modbus, or TCP traffic).
9. **Proxmark3** (RFID Attack Hardware Tool)  
   Hardware-based tool used for emulating, cloning, or replaying RFID access tags.
10. **Mockoon** (Local API Simulation Tool)  
    Emulates REST APIs for wearables or health devices, returning crafted data such as fake heart rate or stress levels.

**Network Tools**

1. Routers
2. USE Wi-Fi Dongle
3. Switches
4. Power board

**Architecture:**

**Cluster 1: Control Room Devices /** **Grid Control Center**

* **Devices**:  
  Smart Cameras (surveillance),  
  RFID Tag Readers (access control),  
  Raspberry Pi (sensor aggregator node),  
  Smart Bulb (facility lighting control),  
  Alexa/Voice Interface (simulate human interaction)
* **Purpose**: Simulate security surveillance, access control, facility environmental control, and human-operator interaction inside the critical infrastructure control room.
* **Attack Simulation**: Camera feed spoofing, Fake RFID access attempts, Ambiguous or repeated Alexa voice commands, Smart bulb flickering

**Cluster 2: Field Sensor Network / Remote Grid Monitoring Stations**

* **Devices**:  
  DHT11 Sensors (temperature/humidity monitoring),  
  MQ2 Sensors (gas leakage detection),  
  Smart Bulbs (remote lighting simulation),  
  Raspberry Pi Pico W (field node connector (smart meter, remote sensor node))
* **Purpose**: Simulate environmental and utility system monitoring in field locations within critical infrastructure.
* **Attack Simulation**: Sensor flooding, Low and slow drift, Remote light control spoofing

**Cluster 3: Human Monitoring and Support/ Grid Operator Health Monitoring**

* **Devices**:  
  Smartwatch (staff motion and fatigue monitoring),  
  Biobeat / Oura Ring (stress, heart rate, sleep tracking),  
  ESP32-CAM (staff surveillance/monitoring),  
  Raspberry Pi Pico W (connect wearables (aggregators)
* **Purpose**: Monitor the physical and cognitive health status of critical infrastructure staff and operators to detect fatigue, stress, or other impairments during operation.
* **Attack Simulation**: Fake high heart rate / stress injection from wearables, False fatigue/alarm signals, Camera feed lag or false movement alerts

A diagram of a cloud server

AI-generated content may be incorrect.

Purchase Link

* + 1. Raspberri pi (15 devices (purchase 4 with at least 8 GB RAM and rest is Ok if 4 GB))-- <https://a.co/d/fTiXTVR>
    2. Pico W (30 devices)--- <https://a.co/d/8I7Jj9V>, <https://www.ebay.ca/itm/144650781896?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=q03azQxXSWC&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    3. Smart cameras (4 devices)-- <https://a.co/d/b8z9QoB>
    4. RFID tag (3 devices) --- <https://a.co/d/hoTh2V6>, <https://a.co/d/5f0qL1i>
    5. Smart bulb (3 packs) --- https://a.co/d/6mX8vm1 , <https://www.ebay.ca/itm/355355800972?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=ngca7kpktqw&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    6. Alexa (2 devices)--- <https://a.co/d/7FxMZ1G>, <https://a.co/d/gLDbA1Q>
    7. Google home (1) -- <https://www.ebay.ca/itm/388317340940?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=HKioi4aaQRC&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    8. DHT11 sensor (10 sensors)-- <https://a.co/d/fgxj1wG>,

<https://www.ebay.ca/itm/314587270631?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=sA1OmmXrT7u&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>,

* + 1. MQ2 sensor (10 sebsors) -- <https://a.co/d/9i1PStg>, <https://www.ebay.ca/itm/185204529905?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=4bFim2W-TcC&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    2. Smart ring (Min 2) -- <https://a.co/d/9PFhRNz>, <https://a.co/d/eRrGWL9>, <https://www.ebay.ca/itm/297248218339?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=7LN-MLDSQbC&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    3. ESP 32 CAM ( 2) -- <https://a.co/d/5DDl3px>, <https://www.ebay.ca/itm/386196331590?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=h4ihfer-rx2&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>
    4. NVIDIA jestson nano board (1 or 2)-- <https://a.co/d/8lPMjgp>, <https://a.co/d/a6t9zSr>
    5. Zigbee (1 or 2) (Please double check this as I never bought Zigbee before so) -- <https://a.co/d/jaGul5T>, <https://a.co/d/5QnkTdL>
    6. Zigbee Devices - <https://a.co/d/6WW392S>,
    7. Wifi Router (1, if we don’t have) -- <https://a.co/d/gVGcyA9>, <https://a.co/d/acmyRpm>
    8. Breadboard to Jumper wire -- <https://a.co/d/aOdmyqj>
    9. Breadboard and jumper kits-- <https://a.co/d/bRrBFOT>

2) Network Topology Setup

1. Divide testbed into **3 clusters as mentioned above.**

**2. Setup Router**

**3. Assign static IPs to devices**

1. Using Router

2. Using DHCP configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Device Name** | **Static IP Possible?** | **Method/Config File** | **Example Command or Setting** |
| Raspberry Pi | Yes | dhcpcd.conf | interface wlan0 (replace this with actual interface) static ip\_address=192.168.1.101/24 gateway=192.168.1.1 (this just example IP and gateway) |
| Raspberry Pi Pico W | Yes | boot.py (MicroPython) | import network wlan = network.WLAN(network.STA\_IF) wlan.active(True) wlan.ifconfig(('192.168.1.201', '255.255.255.0', '192.168.1.1', '8.8.8.8')) |
| NVIDIA Jetson Nano | Yes | dhcpcd.conf | interface eth0  static ip\_address=192.168.1.121/24  gateway=192.168.1.1 |
| ESP32-CAM | Yes | MicroPython | WiFi.config(IPAddress(192,168,1,211), gateway, subnet) |

**DHT11 / MQ2 Sensors**

They are connected **to a host device** (like Raspberry Pi or Pico W) via GPIO pins.

**Smart Bulbs, Smart Cameras, Alexa, Google Home**

* Let them **connect automatically** via the app.
* Then, go to our router’s admin panel **→ DHCP settings → Bind each MAC address to a fixed IP** .

**RFID Tags & Smart Rings**

* Connect Tag with RFID reader and assign IP to reader if not possible then connect it with Raspberry pi
* To connect smart rings with networks, connect with its application to cloud and then cloud to Pi (we need to check how it is then will do)

3) Install and Configure Tools

* Install Zeek, Wireshark, tcpdump, and Suricata on Raspberry Pi (Edge) and Jetson Nano
* Install Python and python libraries on Edge devices
* Set up local protocols: Install Mosquitto MQTT broker on Pi

Core Python Libraries

|  |  |  |
| --- | --- | --- |
| **Library** | **Purpose** | **Installation Command** |
| numpy | Numerical operations | pip install numpy |
| pandas | Data handling & analysis | pip install pandas |
| scikit-learn | Traditional ML models (SVM, RF, KNN, etc.) | pip install scikit-learn |
| matplotlib | Data visualization | pip install matplotlib |
| seaborn | Advanced visualization | pip install seaborn |
| tensorflow / tensorflow-lite | Deep learning framework (standard / lightweight for Pi) | pip install tensorflow |
| torch | PyTorch deep learning framework | pip install torch |
| torchvision | Computer vision utilities for PyTorch | pip install torchvision |
| onnxruntime | Run ONNX models on edge devices | pip install onnxruntime |
| keras | High-level API for building models (with TensorFlow) | pip install keras |
| shap | Explainable AI - SHAP values | pip install shap |
| lime | Explainable AI - local explanations | pip install lime |
| flask | Serve models via lightweight web API | pip install flask |
| paho-mqtt | MQTT communication for IoT devices | pip install paho-mqtt |
| bleak | Bluetooth device interaction (e.g., smart rings) | pip install bleak |
| opencv-python | Computer vision/image input processing | pip install opencv-python |
| psutil | Resource monitoring (CPU, RAM, etc.) | pip install psutil |

**Cluster-1 (Control Room)**

**Install Ubuntu 22.04 LTS (64-bit) for Raspberry Pi**. (<https://ubuntu.com/download/raspberry-pi>)

|  |  |  |  |
| --- | --- | --- | --- |
| Raspberry Pi | IP Address (example) | Assigned Role | Main Functions |
| **Pi-1** | 192.168.1.10 | **Control Aggregator** | Collect data from Pico W (sensors), store logs, relay to Pi-3 |
| **Pi-2** | 192.168.1.11 | **Device Interface Hub** | Interface with smart bulbs, RFID readers, and Alexa |
| **Pi-3** | 192.168.1.12 | **Local Server & Dashboard** | Acts as lightweight server (Flask or MQTT), hosts monitoring dashboard |

* **Network Setup – All Pis connect to your lab router**
* Configure static IPs via /etc/dhcpcd.conf (or bind MAC in router)
* Example config (on Pi): (Do for all three Raspberry pi)

sudo nano /etc/dhcpcd.conf

interface wlan0

static ip\_address=192.168.1.10/24

static routers=192.168.1.1

static domain\_name\_servers=8.8.8.8

Also change IP address and port number in the all given code as per the actual IP address and port number.

* **Pi-1 - Control Aggregator Node**

On Pi-1 execute below commands

sudo apt install python3-dev build-essential

sudo apt update && sudo apt install -y python3-pip tcpdump

pip3 install flask pandas numpy paho-mqtt

This pi collect data from the 10 Raspberry pi Pico W (HTTP and MQQT)🡪

Now first connect all the Pico W with the laptop and manually use below code to connect with router and send simulated data to pi-1.

Here I provided three simulated code so you can divide in a 4,3,3 picos

Three complete simulation setups for Raspberry Pi Pico W using MicroPython to mimic critical infrastructure data (Power Grid Sensor, Water Flow Sensor, and Gas Leak Detector), and corresponding code for:

* **Pi-1**: Aggregator (receives from Pico W via HTTP or MQTT)
* **Pi-3**: Logger server

**Setup 1: Power Grid Voltage Sensor (HTTP POST)**

**Pico W Code (MicroPython) – power\_grid\_sensor.py**

import network

import urequests

import utime

import random

SSID = 'YourWiFiSSID'

PASSWORD = 'YourWiFiPassword'

PI1\_IP = '192.168.1.10'

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():

utime.sleep(1)

print("Connected to WiFi")

while True:

voltage = round(220 + random.uniform(-5, 5), 2)

payload = {"device": "power\_grid\_sensor", "value": voltage}

try:

response = urequests.post(f"http://{PI1\_IP}:5001/data", json=payload)

print("Sent:", response.status\_code)

except Exception as e:

print("Failed:", e)

utime.sleep(5)

**Setup 2: Water Flow Meter (MQTT)**

Pico W Code (MicroPython) **– water\_flow\_mqtt.py**

import network

import time

import random

from umqtt.simple import MQTTClient

SSID = 'YourWiFiSSID'

PASSWORD = 'YourWiFiPassword'

PI1\_IP = '192.168.1.10'

CLIENT\_ID = "flow\_meter"

TOPIC = b"infra/water\_flow"

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():

time.sleep(1)

print("Connected to WiFi")

client = MQTTClient(CLIENT\_ID, PI1\_IP)

client.connect()

while True:

flow\_rate = round(10 + random.uniform(0, 2), 2)

msg = f"{flow\_rate}"

client.publish(TOPIC, msg)

print("Published:", msg)

time.sleep(5)

**Setup 3: Gas Leak Detector (HTTP GET Simulation)**

Pico W Code (MicroPython) **– gas\_detector.py**

import network

import urequests

import utime

import random

SSID = 'YourWiFiSSID'

PASSWORD = 'YourWiFiPassword'

PI1\_IP = '192.168.1.10'

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():

utime.sleep(1)

print("Connected to WiFi")

while True:

gas\_level = random.randint(300, 600)

try:

urequests.get(f"http://{PI1\_IP}:5001/gas?level={gas\_level}")

print("Sent gas level:", gas\_level)

except:

print("Error sending gas level")

utime.sleep(10)

**Pi-1 Aggregator Code (Handles HTTP + MQTT), now on pi-1 write and save below code**

from flask import Flask, request

import paho.mqtt.client as mqtt

import requests

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

PI3\_SERVER = "http://192.168.1.12:5000/log"

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

def receive\_data():

data = request.get\_json()

print("HTTP Data:", data)

requests.post(PI3\_SERVER, json=data)

return "Received", 200

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

def receive\_gas():

gas = request.args.get('level')

payload = {"device": "gas\_detector", "status": f"Gas level: {gas}"}

print(payload)

requests.post(PI3\_SERVER, json=payload)

return "Logged", 200

# MQTT Bridge

def on\_message(client, userdata, msg):

payload = {"device": msg.topic, "status": msg.payload.decode()}

print("MQTT Data:", payload)

requests.post(PI3\_SERVER, json=payload)

mqtt\_client = mqtt.Client()

mqtt\_client.on\_message = on\_message

mqtt\_client.connect("localhost", 1883)

mqtt\_client.subscribe("infra/#")

mqtt\_client.loop\_start()

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

app.run(host='0.0.0.0', port=5001)

* **Pi-2: Interface Hub for Smart Devices (I never tried it so do it carefully)**

**Install below software on it**

sudo apt update && sudo apt install -y python3-pip nmap

pip3 install flask gpiozero paho-mqtt

Connect below Devices router

 **Smart bulbs** (via MQTT or HTTP API)🡪 4 bulbs

 **RFID Readers** (via GPIO or serial port)🡪 2 Readers

 **Alexa/Google Home** (through local cloud relay or MQTT actions)🡪 2 Alexa

**This pi send data via HTTP or MQTT to Pi 3 which is Local Server & Dashboard**

* **Connect bulb to Wi-Fi**

### **Step 1: Confirm Tuya App and Account**

1. Download **Smart Life** or **Tuya Smart** app
2. Pair all 4 bulbs in the app
3. Create a Tuya IoT developer account: <https://iot.tuya.com/cloud/>

Now allowing **Pi-2** (through the tuya-mqtt bridge) to control Tuya-based smart bulbs **locally via MQTT**, without using their official app only.

### **STEP 1: Create Tuya IoT Developer Account**

1. **Go to: https://iot.tuya.com/**
2. Click **Sign Up** (if new) or **Login** (if already registered)
   * Use the **same email or phone number** that you used in the **Smart Life app**

### **STEP 2: Create a Cloud Project**

1. After login, go to the **Cloud > Development** section
   * Or click this shortcut: https://iot.tuya.com/cloud/
2. Click **“Create Cloud Project”**
3. Fill in the details.

### **STEP 3: Add APIs to Your Project**

1. After creating the project, go to **“API Group Management”**
2. Click **"Authorize APIs"**
3. Select:
   * **Device Control**
   * **Device Management**
   * **User Management**
   * **Smart Home Scene Linkage**
   * **MQTT Device Communication**
4. Click **Confirm** to authorize

### **STEP 4: Link Your Smart Life App Account (Important)**

Now **link Tuya App account** (where bulbs are already set up) with Tuya IoT Cloud project.

1. Go to **Tuya Cloud Project → Devices → Link Tuya App Account**
2. Click **“Add App Account”**
3. Use phone (with Smart Life app installed):
   * Go to **“Me → Settings → Account and Security → Scan QR Code”**
   * Use the Tuya app to **scan the QR code** shown on the Tuya IoT website

Once scanned successfully, it links your cloud project to your real smart devices.

### **STEP 5: Copy Your Credentials**

After account is linked, copy the following from the **Tuya IoT Console → Project Overview**:

* **Access ID**
* **Access Secret**
* **Username** (your email or phone used in Smart Life)
* **Password** (your Smart Life login password)
* **Region/Endpoint** (usually us, eu, or cn)

Connect Tuya Bulbs to Pi-2

Step 1: Install Required Packages on Pi-2

sudo apt update

sudo apt install -y python3-pip git

pip3 install paho-mqtt tinytuya

Step 2: Clone the Tuya-MQTT Bridge

cd ~

git clone https://github.com/TradeFace/tuyamqtt.git

cd tuyamqtt

Step 3: Create the Configuration File

**nano config.json**

Paste the following in the json file and **fill in your own credentials** from the Tuya Cloud Console:

**{**

**"mqtt": {**

**"host": "localhost",**

**"port": 1883,**

**"topic\_prefix": "tuya/",**

**"retain": false**

**},**

**"tuya": {**

**"access\_id": "YOUR\_TUYA\_ACCESS\_ID",**

**"access\_secret": "YOUR\_TUYA\_SECRET",**

**"username": "YOUR\_SMART\_LIFE\_EMAIL",**

**"password": "YOUR\_SMART\_LIFE\_PASSWORD",**

**"country\_code": "1",**

**"region": "us"**

**}**

**}**

Step 4: Run the MQTT Bridge

Now, start the script to bridge MQTT with Tuya:

**python3 main.py**

output like:

**Connected to Tuya cloud**

**MQTT server listening at localhost:1883**

**Device registered: bulb1**

### **Step 5: Control a Bulb via MQTT**

From another terminal or another device on the same network:

**Turn ON:**

**mosquitto\_pub -h 192.168.1.11 -t "tuya/bulb1/command" -m "on"**

Turn OFF:

**mosquitto\_pub -h 192.168.1.11 -t "tuya/bulb1/command" -m "off"**

**Now write below code on pi-2 and save it ( mqtt\_to\_http\_forwarder.py)**

import paho.mqtt.client as mqtt

import requests

def on\_message(client, userdata, msg):

data = msg.payload.decode()

requests.post("http://192.168.1.12:5000/log", json={

"device": msg.topic,

"status": data

})

client = mqtt.Client()

client.on\_message = on\_message

client.connect("localhost", 1883, 60)

client.subscribe("tuya/#") # or specific like tuya/bulb1/status

client.loop\_forever()

Run it :

python3 mqtt\_to\_http\_forwarder.py

**Flask Server Code (run on Pi-3)**

# pi3\_logger\_server.py

from flask import Flask, request

import pandas as pd

import os

from datetime import datetime

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

LOG\_PATH = "/home/pi/data/bulb\_log.csv"

# Initialize log file

if not os.path.exists(LOG\_PATH):

df = pd.DataFrame(columns=["timestamp", "device", "status"])

df.to\_csv(LOG\_PATH, index=False)

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

def log\_event():

data = request.get\_json()

if not data:

return "No data received", 400

timestamp = datetime.now().isoformat()

device = data.get("device", "unknown")

status = data.get("status", "unknown")

df = pd.DataFrame([[timestamp, device, status]], columns=["timestamp", "device", "status"])

df.to\_csv(LOG\_PATH, mode='a', header=False, index=False)

print(f"Logged: {device} → {status}")

return "Logged", 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=5000)

**Run this code :**

python3 pi3\_logger\_server.py

We Can also run this as a background service so just check that.

* **Now connect RFID to pi-2 and then send data to pi-3**

**Step 1:** **Hardware Wiring (RC522 to Pi-2) --- this for one reader**

|  |  |  |
| --- | --- | --- |
| RC522 Pin | Raspberry Pi GPIO Pin | BCM Pin |
| SDA | GPIO 8 (CE0) | 24 |
| SCK | GPIO 11 (SCLK) | 23 |
| MOSI | GPIO 10 (MOSI) | 19 |
| MISO | GPIO 9 (MISO) | 21 |
| GND | GND | 6 |
| RST | GPIO 25 | 22 |
| 3.3V | 3.3V | 1 |

**Step-2** Install below on pi-2

pip3 install RPi.GPIO spidev mfrc522 requests

**Enable SPI:**

sudo raspi-config

# Go to: Interface Options → SPI → Enable → Reboot if asked

**Test SPI is enabled:**

ls /dev/spidev0.\*

# Output should be: /dev/spidev0.0 and /dev/spidev0.1

**Step 3:** **Create Script to Read RFID and Send to Pi-3**

Save the below Python code on **Pi-2** as rfid\_to\_http\_forwarder.py:

from mfrc522 import SimpleMFRC522

import requests

from datetime import datetime

import time

reader = SimpleMFRC522()

SERVER\_URL = "http://192.168.1.12:5000/log" # Pi-3 Flask server

print("Place your RFID tag near the reader...")

try:

while True:

id, text = reader.read()

timestamp = datetime.now().isoformat()

print(f"Tag ID: {id}, Text: {text.strip()}")

payload = {

"device": "rfid\_reader\_1",

"status": f"ID: {id}, Text: {text.strip()}"

}

try:

res = requests.post(SERVER\_URL, json=payload)

print("Sent to server:", res.status\_code)

except Exception as e:

print("Failed to send:", e)

time.sleep(2) # avoid multiple scans in rapid succession

except KeyboardInterrupt:

print("Stopped by user")

finally:

GPIO.cleanup()

Run it:

python3 rfid\_to\_http\_forwarder.py

To avoid logging the **same tag twice in a row**, you can track the last tag ID:

last\_id = None

...

if id != last\_id:

# send request

last\_id = id

**Step-4: Flask Server on Pi-3 (Already Created)**

python3 pi3\_logger\_server.py

We Can also run this as a background service so just check that.

* Connect Alexa with pi-2

**Step1:**

1. Sign up at: https://developer.amazon.com/alexa
2. Create a **custom skill** (with invocation name like "Smart Lab")
3. Define **intents**, e.g., TurnOnLightIntent
4. Use **Lambda function** to: (This is just a small we can do it complex and dynamic)

import requests

def lambda\_handler(event, context):

requests.post("http://<Pi-2-IP>:5000/alexa", json={"command": "light on"})

return {

'version': '1.0',

'response': {

'outputSpeech': {

'type': 'PlainText',

'text': 'Okay, turning on the light.'

},

'shouldEndSession': True

}

}

Step-2 Flask server on pi-2

Save this as alexa\_listener.py on **Pi-2**:

from flask import Flask, request

import requests

from datetime import datetime

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

PI3\_SERVER\_URL = "http://192.168.1.12:5000/log" # Pi-3 endpoint

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

def receive\_alexa\_command():

data = request.get\_json()

command = data.get("command", "none")

print(f"Received Alexa command: {command}")

# Forward to Pi-3

payload = {

"device": "alexa",

"status": command

}

try:

res = requests.post(PI3\_SERVER\_URL, json=payload)

print("Forwarded to Pi-3:", res.status\_code)

except Exception as e:

print("Failed to forward:", e)

return "Command processed", 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=5000)

Run it

python3 alexa\_listener.py

**Step-3: Flask Server on Pi-3 (Already Created)**

python3 pi3\_logger\_server.py

* **Pi-3: Local Monitoring Server (Dashboard + Logger)**

**Install below software on this pi**

If you encounter any issues with Zeek installation, refer to the link below. It is slightly outdated but still useful

<https://medium.com/@cybertoolguardian/zeek-installation-in-ubuntu-60835ee3e42c>

sudo apt update && sudo apt install -y \

python3-pip \

mosquitto \

mosquitto-clients \

tcpdump \

wireshark \

zeek \

git \

nmap \

curl

pip3 install flask streamlit pandas matplotlib psutil

sudo apt install logrotate

sudo apt install influxdb

If you encounter any issues with Zeek installation, refer to the link below. It is slightly outdated but still useful

<https://medium.com/@cybertoolguardian/zeek-installation-in-ubuntu-60835ee3e42c>

|  |  |
| --- | --- |
| Tool | Role in Critical Infrastructure Monitoring |
| **Zeek** | Network-level logger (extracts HTTP, TLS, DNS, MQTT if custom plugin added) |
| **Wireshark** | Manual packet inspection (real-time or offline .pcap viewer) |
| **tcpdump** | Raw capture of .pcap files |
| **Mosquitto** | MQTT broker to receive data from Pico or Pi-1 |
| **Flask / Streamlit** | Lightweight HTTP-based dashboard or interactive charts |
| **psutil** | Monitor CPU/memory to simulate resource usage at edge |
| **nmap** | Optional internal network scan tool to validate connected devices |
| **curl** | Used to receive/send test data from endpoints |