Contexto

A organização COMCS.Lda é uma empresa que vende uma gama de produtos manufaturados nas suas unidades industriais e guardadas num armazém centralizado. Dada a sensibilidade dos seus produtos, é necessário garantir que as diferentes etapas de produção, armazenamento e transporte mantenham níveis constantes de temperatura e humidade. Pequenas variações podem comprometer a qualidade dos produtos. Identificar a origem dos problemas é fulcral para assegurar qualidade e a satisfação dos clientes. De modo a responder rapidamente a esta necessidade, é preciso desenvolver uma solução que monitorize os indicadores mencionados, i.e., temperatura e humidade relativa.

Estrutura da Solução

A Figura 1 é uma representação da estrutura geral da solução proposta.

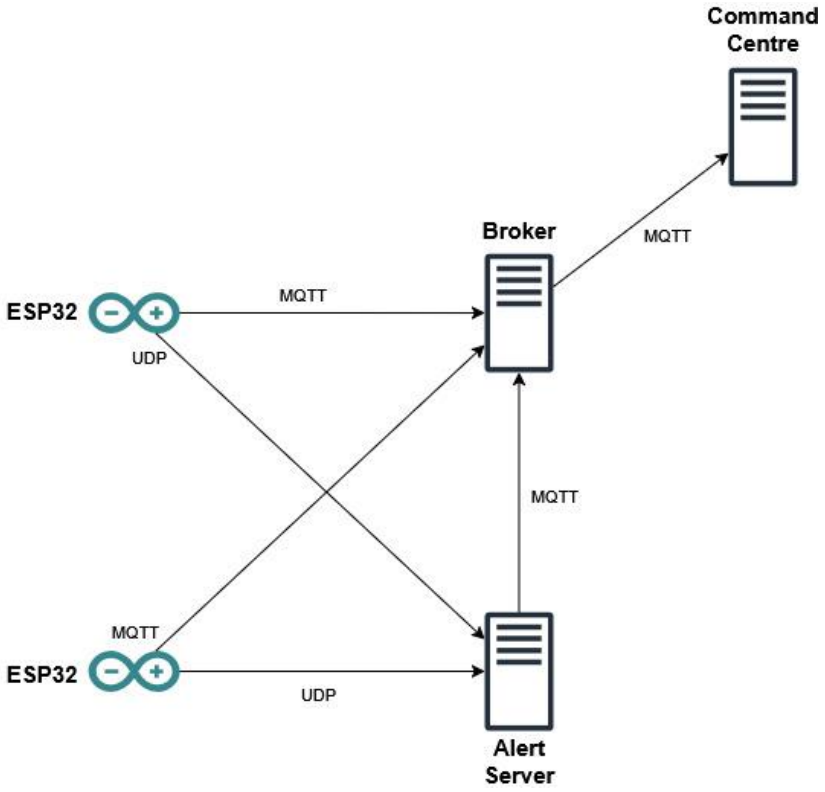

Figura 1 – Estrutura da solução proposta.

ESP32

O ESP32 faz as medições de temperatura e a humidade do ambiente a partir do sensor DHT11. Depois, o Arduino constrói uma estrutura que aglomera tais dados de uma forma compacta no formato JSON, de acordo com os conformes da proposta *Smart Data Models* proposta por Ahmed Abid em https://github.com/smart-data-models/dataModel.Weather/blob/master/WeatherObserved/doc/spec.md . A Figura 2 mostra um exemplo de uma medição bem-sucedida do ESP32.


```
{
  "id": "WeatherObserved:ESP_TRH01:2025-12-02T13:22:41.00Z",
  "dateObserved": "2025-12-02T13:22:41.00Z",
  "location": {
    "type": "Point",
    "coordinates": [-8.60622, 41.17878]
  },
  "type": "WeatherObserved",
  "temperature": 15.7,
  "relativeHumidity": 74
}
```


Figura 2 - Estrutura de dados enviada pelo ESP32

Uma vez construída, a mensagem é enviada tanto para o **Broker** quanto para o **Alert Server**. No caso do Broker, esta publicação é feita no tópico publica no tópico "/comcs/sensor" utilizando o endereço e a porta do servidor; já no caso do Alert Server, basta o endereço e o port do servidor. No caso do UDP server foi implementada a funcionalidade de QoS 1, que pode ser estabelecido através da variável global `qos_udp`. Este QoS permite ao ESP32 verificar se o Alert Server recebeu a estrutura de dados e, caso não tenha sido entregue, reenvia até os dados serem recebidos pelo servidor. Contudo, dado que novas medições de temperatura e humidade são feitas a cada segundo, é também estabelecido um `timeout` ao fim do qual o ESP32 para de tentar receber a mensagem de ACK "UDP Received", reenvia a mensagem novamente e passa para a próxima medição.

Para cumprir o ponto 4 do enunciado, foi implementada uma janela deslizante de estado no ESP32 recorrendo ao sistema de ficheiros SPIFFS. O objetivo é garantir que, em caso de falha de comunicação (MQTT ou UDP) ou de reinício do dispositivo, as últimas medições não são perdidas e são reenviadas para o servidor UDP (Alert Server) e para o Command Centre (via MQTT) assim que possível.

Nota: O ID de cada estrutura é definido como **WeatherObserved:[esp_id]:[date-time].** O ID do ESP32 foi definido estaticamente como variável global (`esp_id`).

Configuração

ESP32

Ambos os dispositivos ESP32 devem ser posicionados em locais estratégicos da sala para cobrir a maior área possível. Caso estejam demasiado próximos, os valores medidos podem ser praticamente idênticos, sendo difícil saber se existem variações grandes de temperatura ou humidade na sua área de atuação.

Etapas de configuração do ESP32:

1. Instalar o Arduino IDE.
2. Adicionar suporte para ESP32 ao Arduino IDE.
    a. Tools > Board > Board Manager e instalar a board **esp32**, por Expressif.
    b. Selecionar a board instalada no menu Board Manager.
3. Instalar as bibliotecas necessárias:
    a. **DHT Sensor Library**, por adafruit.
    b. **PubSubClient**, por Nick O'Leary.
4. Carregar o código para o ESP32:
    - Ligar o ESP32 ao computador via cabo USB.
    - Configurar todos os parâmetros necessários:
        - Credenciais de WiFi.
        - Credenciais do Broker MQTT, bem como o endereço e port.
        - Endereço e port do Alert Server.
        - Identificador único do ESP32.
        - Tamanho dos logs, e nome do ficheiro onde são guardados.
    - Carregar o código para o dispositivo através do botão Upload.

Servidor UDP

Setup

Instalar paho.mqtt.c

git clone https://github.com/eclipse-paho/paho.mqtt.c

make

make install

Instalar cJSON

git clone https://github.com/DaveGamble/cJSON

make

make install

Compilar

gcc -o server udp_server.c -lpaho-mqtt3cs -lcjson –pthread

Correr o servidor

./server

gcc -o tests udp_tests.c -lpthread

Correr testes (clientes, número de mensagens, host, port)

./tests 20 100 192.168.64.2 8080

Descrição

Limites para temperatura: min 0, max 50

Limites para humidade: min 20, max 80

Limites para o diferencial: temperatura 2, humidade 5

Para correr os testes usar o udp_tests (./tests se for compilado como a recomendação), para um teste simples pode ser usado echo
echo "{\"id\":\"esp\",\"type\":\"WeatherObserved\",\"dateObserved\":\"2016-11-30T07:00:00.00Z\",\"location\":{\"type\":\"Point\",\"coordinates\":[-4.754444444,41.640833333]},\"relativeHumidity\":\"60\",\"temperature\":\"20\"}" | nc -u -w1 <host> <port>

Os testes enviam X mensagens por X clients, com 4 mensagens por client que dão trigger aos 4 alerts.

O diferencial é calculado com uma média incremental, desta forma não só são detetadas discrepâncias entre os dois ESP32 como também no mesmo.

O servidor usa QoS 2 para o broker MQTT para se certificar que os alerts chegam.

O servidor devolve também uma mensagem ao client (ESP32) de acknowledgment.

Command Centre (Node-Red)

Setup

Correr o comando “node-red”

Abrir o browser, por exemplo http://192.168.1.206:1880/

Importar a flow (flow.json)

Configurar o broker MQTT

Deploy

Abrir o browser para o UI, http://192.168.1.206:1880/ui