Framework Arduino

#include <WiFi.h>

#include <MQTTPubSubClient.h>

#include <Wire.h>

#include <SPI.h>

#include <Adafruit\_BMP280.h>

Adafruit\_BMP280 bme;

float temperatura, umidade, pressao, altitude;

const char\* ssid = "JMJ\_2.4GHz";

const char\* pass = "06071012";

WiFiClient client;

MQTTPubSubClient mqtt;

IPAddress ip(192, 168, 1, 14);

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, pass);

Serial.println("connecting to BMP...");

unsigned status;

status = bme.begin(0x76);

while (!status) {

Serial.print(".");

delay(200);

}

Serial.print("connecting to wifi...");

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(1000);

}

Serial.println(" connected!");

Serial.print("connecting to host...");

while (!client.connect(ip, 1883)) {

Serial.print(".");

delay(1000);

}

Serial.println(" connected!");

// initialize mqtt client

mqtt.begin(client);

Serial.print("connecting to mqtt broker...");

while (!mqtt.connect("arduino", "admin", "1234")) {

Serial.print(".");

delay(1000);

}

Serial.println(" connected!");

// subscribe callback which is called when every packet has come

mqtt.subscribe([](const String& topic, const String& payload, const size\_t size) {

Serial.println("mqtt received: " + topic + " - " + payload);

});

// subscribe topic and callback which is called when /hello has come

mqtt.subscribe("/hello", [](const String& payload, const size\_t size) {

Serial.print("/hello ");

Serial.println(payload);

});

}

void loop() {

mqtt.update(); // should be called

temperatura = bme.readTemperature();

pressao = bme.readPressure()/100;

char buffer[40];

sprintf(buffer, "{\"temp\":%.2f,\"pressure\":%.2f}", temperatura, pressao);

Serial.println(buffer);

mqtt.publish("/estacao", buffer);

delay(500);

}

Framework ESP-IDF

#include <stdio.h>

#include <stdint.h>

#include <stddef.h>

#include <string.h>

#include "esp\_wifi.h"

#include "esp\_system.h"

#include "nvs\_flash.h"

#include "esp\_event.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include <bmp280.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/semphr.h"

#include "freertos/queue.h"

#include "lwip/sockets.h"

#include "lwip/dns.h"

#include "lwip/netdb.h"

#include "esp\_log.h"

#include "mqtt\_client.h"

#include <string.h>

#define I2C\_MASTER\_SDA 21

#define I2C\_MASTER\_SCL 22

static const char \*TAG = "MQTT\_EXAMPLE";

struct dadosClima{

float temperatura;

float pressao;

};

void create\_msg(char \*msg, struct dadosClima \*msg\_struct)

{

sprintf(msg,

"{\"temp\":%.2f,\"pressure\":%.2f}",

msg\_struct->temperatura, msg\_struct->pressao);

}

void bmp280\_test(void \*pvParameters)

{

esp\_mqtt\_client\_handle\_t client\_handler = (esp\_mqtt\_client\_handle\_t) pvParameters;

bmp280\_params\_t params;

bmp280\_init\_default\_params(&params);

bmp280\_t dev;

memset(&dev, 0, sizeof(bmp280\_t));

ESP\_ERROR\_CHECK(bmp280\_init\_desc(&dev, BMP280\_I2C\_ADDRESS\_0, 0, I2C\_MASTER\_SDA, I2C\_MASTER\_SCL));

ESP\_ERROR\_CHECK(bmp280\_init(&dev, &params));

bool bme280p = dev.id == BME280\_CHIP\_ID;

printf("BMP280: found %s\n", bme280p ? "BME280" : "BMP280");

float pressure, temperature, humidity;

while (1)

{

vTaskDelay(pdMS\_TO\_TICKS(500));

if (bmp280\_read\_float(&dev, &temperature, &pressure, &humidity) != ESP\_OK)

{

printf("Temperature/pressure reading failed\n");

continue;

}

struct dadosClima s1;

s1.temperatura = temperature;

s1.pressao = pressure/100.0;

char json\_msg[100];

create\_msg(json\_msg, &s1);

esp\_mqtt\_client\_publish(client\_handler,"/estacao" , json\_msg, 0,0,0);

printf("->>>> %s",json\_msg);

if (bme280p)

printf(", Humidity: %.2f\n", humidity);

else

printf("\n");

}

}

static void log\_error\_if\_nonzero(const char \* message, int error\_code)

{

if (error\_code != 0) {

ESP\_LOGE(TAG, "Last error %s: 0x%x", message, error\_code);

}

}

static esp\_err\_t mqtt\_event\_handler\_cb(esp\_mqtt\_event\_handle\_t event)

{

int msg\_id;

// your\_context\_t \*context = event->context;

switch (event->event\_id) {

case MQTT\_EVENT\_CONNECTED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_CONNECTED");

break;

case MQTT\_EVENT\_DISCONNECTED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_DISCONNECTED");

break;

case MQTT\_EVENT\_SUBSCRIBED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_SUBSCRIBED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_UNSUBSCRIBED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_UNSUBSCRIBED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_PUBLISHED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_PUBLISHED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_DATA:

ESP\_LOGI(TAG, "MQTT\_EVENT\_DATA");

printf("TOPIC=%.\*s\r\n", event->topic\_len, event->topic);

printf("DATA=%.\*s\r\n", event->data\_len, event->data);

break;

case MQTT\_EVENT\_ERROR:

ESP\_LOGI(TAG, "MQTT\_EVENT\_ERROR");

if (event->error\_handle->error\_type == MQTT\_ERROR\_TYPE\_TCP\_TRANSPORT) {

log\_error\_if\_nonzero("reported from esp-tls", event->error\_handle->esp\_tls\_last\_esp\_err);

log\_error\_if\_nonzero("reported from tls stack", event->error\_handle->esp\_tls\_stack\_err);

log\_error\_if\_nonzero("captured as transport's socket errno", event->error\_handle->esp\_transport\_sock\_errno);

ESP\_LOGI(TAG, "Last errno string (%s)", strerror(event->error\_handle->esp\_transport\_sock\_errno));

}

break;

default:

ESP\_LOGI(TAG, "Other event id:%d", event->event\_id);

break;

}

return ESP\_OK;

}

static void mqtt\_event\_handler(void \*handler\_args, esp\_event\_base\_t base, int32\_t event\_id, void \*event\_data) {

ESP\_LOGD(TAG, "Event dispatched from event loop base=%s, event\_id=%d", base, event\_id);

mqtt\_event\_handler\_cb(event\_data);

}

static esp\_mqtt\_client\_handle\_t mqtt\_app\_start(void)

{

esp\_mqtt\_client\_config\_t mqtt\_cfg = {

.username = "admin",

.password = "1234",

.uri = "mqtt://192.168.1.14",

};

esp\_mqtt\_client\_handle\_t client = esp\_mqtt\_client\_init(&mqtt\_cfg);

esp\_mqtt\_client\_register\_event(client, ESP\_EVENT\_ANY\_ID, mqtt\_event\_handler, client);

esp\_mqtt\_client\_start(client);

return client;

}

void app\_main(void)

{

esp\_mqtt\_client\_handle\_t client\_handler;

ESP\_LOGI(TAG, "[APP] Startup..");

ESP\_LOGI(TAG, "[APP] Free memory: %d bytes", esp\_get\_free\_heap\_size());

ESP\_LOGI(TAG, "[APP] IDF version: %s", esp\_get\_idf\_version());

ESP\_ERROR\_CHECK(nvs\_flash\_init());

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

ESP\_ERROR\_CHECK(i2cdev\_init());

ESP\_ERROR\_CHECK(example\_connect());

client\_handler = mqtt\_app\_start();

xTaskCreatePinnedToCore(bmp280\_test, "bmp280\_test", configMINIMAL\_STACK\_SIZE \* 8, (void \*)client\_handler, 5, NULL, APP\_CPU\_NUM);

}