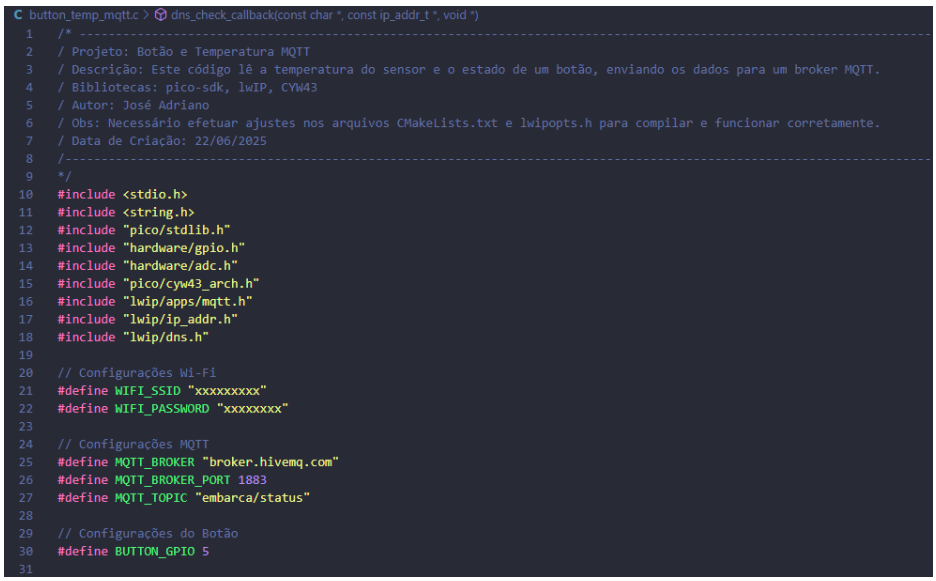
Aplicação com comunicação sem fio para IoT

Nome: **José Adriano Filho**
Matrícula: **2025101109806**
Unidade2-Capítulo3

1. Utilizando a placa Bitdoglab, crie um programa para monitorar o status de um pino, que possa ser definido como entrada, da placa Bitdoglab, e enviar, a cada 1 segundo o status atual para o servidor HiveMQ, utilizando o protocolo MQTT. Além disso, como desafio extra, acrescente algum sensor e envie a informação desse sensor para o servidor.

R.: link do código:
https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/button_temp_mqtt

O código foi desenvolvido baseado em exemplos encontrados na internet já que existe uma dificuldade de documentações sobre MQTT em linguagem C. Procuramos desenvolver o monitoramento do botão e a leitura do sensor de temperatura interno, sendo que a cada um segundo enviamos o status e a medida em graus centigrados para o broker MQTT na plataforma da HiveMQ.




```c
/* ------------------------------------------------------------------------------------------------------------------
 * Projeto: Botão e Temperatura MQTT
 * Descrição: Este código lê a temperatura do sensor e o estado de um botão, enviando os dados para um broker MQTT.
 * Bibliotecas: pico-sdk, lwIP, CYW43
 * Autor: José Adriano
 * Obs: Necessário efetuar ajustes nos arquivos CMakeLists.txt e lwipopts.h para compilar e funcionar corretamente.
 * Data de Criação: 22/06/2025
 *------------------------------------------------------------------------------------------------------------------
 */
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "pico/cyw43_arch.h"
#include "lwip/apps/mqtt.h"
#include "lwip/ip_addr.h"
#include "lwip/dns.h"

// Configurações Wi-Fi
#define WIFI_SSID "xxxxxxxxx"
#define WIFI_PASSWORD "xxxxxxxx"

// Configurações MQTT
#define MQTT_BROKER "broker.hivemq.com"
#define MQTT_BROKER_PORT 1883
#define MQTT_TOPIC "embarca/status"

// Configurações do Botão
#define BUTTON_GPIO 5

```

Inicialmente fizemos os includes necessários para a comunicação via WiFi, bem como algumas definições importantes.

```c
// Variáveis Globais
static mqtt_client_t *mqtt_client;
static ip_addr_t broker_ip;
static bool mqtt_connected = false;

// Protótipo das Funções
static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
void publish_msg(bool button_pressed, float temp_c);
float read_temperature();
void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg);

```

Acima algumas variáveis importantes e os protótipos das funções necessários para executar a leitura do sensor, a resolução do DNS, ou seja, converter o nome para acesso ao broker em um ip fixo, o callback para conexão MQTT e a função de publicar nossa mensagem no tópico.

A seguir, de acordo com as assinaturas das funções temos os códigoss das funções para conexão do MQTT e publicar o staus do botão e a temperatura interna.

```c
// Callback de conexão MQTT
static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
    if (status == MQTT_CONNECT_ACCEPTED) {
        printf("[MQTT] Conectado ao broker!\n");
        mqtt_connected = true;
    } else {
        printf("[MQTT] Falha na conexão MQTT. Código: %d\n", status);
        mqtt_connected = false;
    }
}

// Publicar botão + temperatura
void publish_msg(bool button_pressed, float temp_c) {
    if (!mqtt_connected) {
        printf("[MQTT] Não conectado, não publicando dados\n");
        return;
    }

    char payload[128];
    snprintf(payload, sizeof(payload),
             "{\"botao\":\"%s\",\"temperatura\":%.2f}",
             button_pressed ? "ON" : "OFF",
             temp_c);

    printf("[MQTT] Publicando: tópico='%s', mensagem='%s'\n", MQTT_TOPIC, payload);

    err_t err = mqtt_publish(mqtt_client, MQTT_TOPIC, payload, strlen(payload), 0, 0, NULL, NULL);
    if (err == ERR_OK) {
        printf("[MQTT] Publicação OK\n");
    } else {
        printf("[MQTT] Erro ao publicar: %d\n", err);
    }
}
```

A seguir as outras duas funções, responsáveis pela leitura da temperatura e a resolução do DNS.

```c
// Leitura da temperatura
float read_temperature() {
    uint16_t raw = adc_read();
    const float conversion_factor = 3.3f / (1 << 12);
    float voltage = raw * conversion_factor;
    float temp_c = 27.0f - (voltage - 0.706f) / 0.001721f;
    return temp_c;
}

// Callback de DNS
void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
    if (ipaddr != NULL) {
        broker_ip = *ipaddr;
        printf("[DNS] Resolvido: %s -> %s\n", name, ipaddr_ntoa(ipaddr));

        struct mqtt_connect_client_info_t client_info = {
            .client_id = "pico-client",
            .keep_alive = 60,
            .client_user = NULL,
            .client_pass = NULL,
            .will_topic = NULL,
            .will_msg = NULL,
            .will_qos = 0,
            .will_retain = 0
        };

        printf("[MQTT] Conectando ao broker...\n");
        mqtt_client_connect(mqtt_client, &broker_ip, MQTT_BROKER_PORT, mqtt_connection_callback, NULL, &client_info);
    } else {
        printf("[DNS] Falha ao resolver DNS para %s\n", name);
    }
}

```

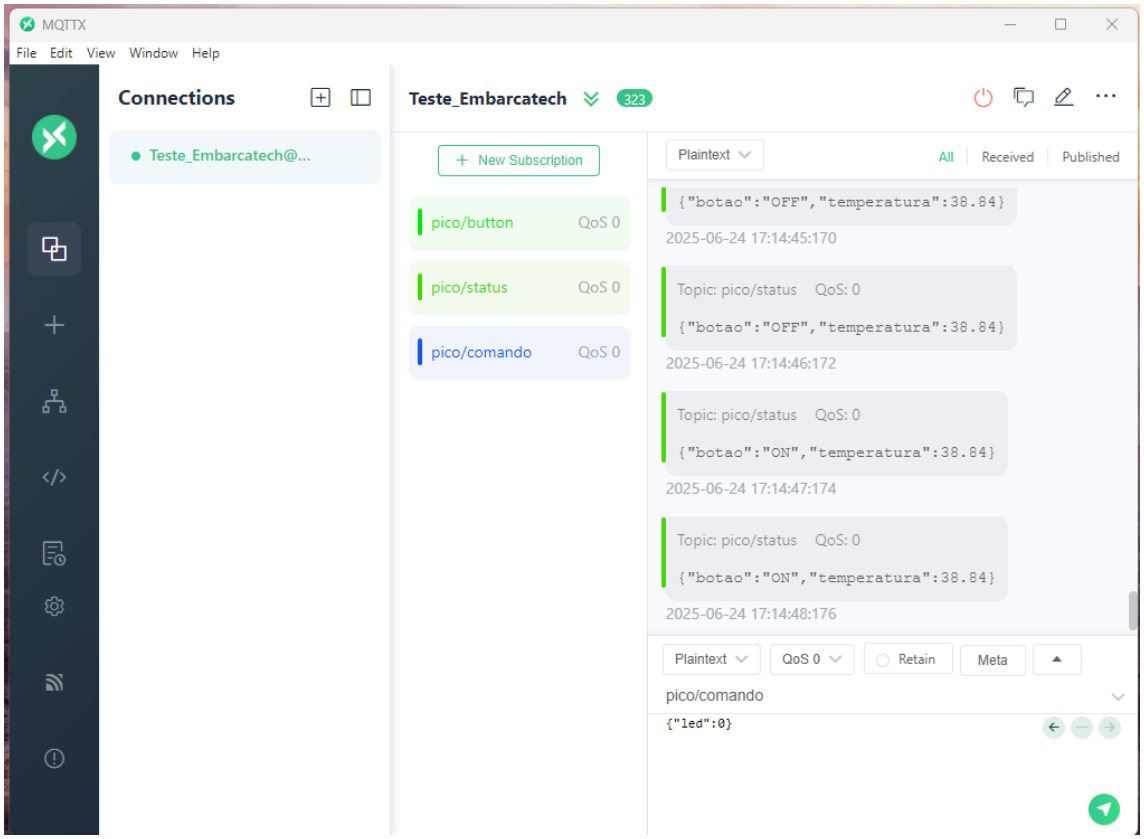
Abaixo o código desenvolvido para a função principal main.

```c
// Função Principal
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("\n=== Iniciando MQTT Button + Temperature ===\n");

    // Inicializa Wi-Fis
    if (cyw43_arch_init()) {
        printf("Erro na inicialização do Wi-Fi\n");
        return -1;
    }
    cyw43_arch_enable_sta_mode();

    printf("[Wi-Fi] Conectando...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
        printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
        return -1;
    } else {
        printf("[Wi-Fi] Conectado com sucesso!\n");
    }

    // Configura GPIO do botão
    gpio_init(BUTTON_GPIO);
    gpio_set_dir(BUTTON_GPIO, GPIO_IN);
    gpio_pull_up(BUTTON_GPIO); // Pull-up ativado

    // Inicializa ADC para temperatura
    adc_init();
    adc_set_temp_sensor_enabled(true);
    adc_select_input(4);

    // Inicializa cliente MQTT
    mqtt_client = mqtt_client_new();

    // Resolve DNS do broker MQTT
    err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_check_callback, NULL);
    if (err == ERR_OK) {
        dns_check_callback(MQTT_BROKER, &broker_ip, NULL);
    } else if (err == ERR_INPROGRESS) {
        printf("[DNS] Resolvendo...\n");
    } else {
        printf("[DNS] Erro ao resolver DNS: %d\n", err);
        return -1;
    }

```

```c
    // Loop principal
    while (true) {
        // Atualiza tarefas de rede
        cyw43_arch_poll();

        // Lê botão
        bool button_state = !gpio_get(BUTTON_GPIO); // Inversão por pull-up

        // Lê temperatura
        float temp_c = read_temperature();
        printf("[TEMP] Temperatura atual: %.2f °C\n", temp_c);

        // Publica ambos no mesmo tópico
        publish_msg(button_state, temp_c);

        // Espera 1 segundo
        sleep_ms(1000);
    }

    cyw43_arch_deinit();
    return 0;
}
```

A cada 1 segundo executamos a atualização do status do botão e do valor do sensor de temperatura.

Os testes foram executados com o cliente MQTT “MQTTX” rodando em um pc e inscrito no tópico “pico/status.



Na saída serial temos alguns “prints” para monitoramento e debug da aplicação.

```
[DNS] Resolvido: broker.hivemq.com -> 3.65.119.144
[MQTT] Conectando ao broker...
[MQTT] Conectado ao broker!
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"OFF","temperatura":41.18}'
[MQTT] Publicação OK
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"ON","temperatura":41.18}'
[MQTT] Publicação OK
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"OFF","temperatura":41.18}'
```

2. Utilizando a placa Bitdoglab, crie um programa para receber dados do servidor HiveMQ, utilizando o protocolo MQTT. Os dados recebidos devem ligar ou desligar um LED, que deve ser ligado a placa. Além disso, como desafio extra, implemente um LCD ou outro tipo de display para monitorar o recebimento de mensagens.

R.: link do código:
https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/led_lcd_mqtt

Nossa proposta agora é comandar um led por comando enviado pelo protocolo MQTT, bem como sinalizar no display da BigDogLab o status dos acontecimentos durante a execução.

src > C led_lcd_mqtt.c > ...

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include "pico/cyw43_arch.h"
#include "lwip/apps/mqtt.h"
#include "lwip/ip_addr.h"
#include "lwip/dns.h"

#include "ssd1306.h"

// ====================== Configurações ========================
#define WIFI_SSID "xxxxxxxxx"
#define WIFI_PASSWORD "xxxxxxxx"

#define MQTT_BROKER "broker.hivemq.com"
#define MQTT_PORT_BROKER 1883

#define MQTT_TOPIC_PUB "embarca/status"
#define MQTT_TOPIC_SUB "embarca/comando"

#define BUTTON_GPIO 5
#define LED_GPIO 12

#define I2C_SDA 14
#define I2C_SCL 15
```

Aqui temos os includes iniciais e as definições necessárias para o desenvolvimento do programa.

```c
// ====================== Variáveis Globais ====================
static mqtt_client_t *mqtt_client;
static ip_addr_t broker_ip;
static bool mqtt_connected = false;
static bool button_last_state = false;

// ====================== Protótipos ===========================
void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len);
void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags);
void publish_button_state(bool pressed);
void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg);
```

Na imagem acima temos as variáveis necessárias, já que nos baseamos em nosso programa anterior, bem como os protótipos das funções desenvolvidas.

Na página seguinte temos a codificação das duas funções.

```c
// ================== Callbacks e Publicação ====================
void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
    if (ipaddr != NULL) {
        broker_ip = *ipaddr;
        printf("[DNS] %s -> %s\n", name, ipaddr_ntoa(ipaddr));

        struct mqtt_connect_client_info_t ci = {
            .client_id = "pico-client",
            .keep_alive = 60,
            .client_user = NULL,
            .client_pass = NULL,
            .will_topic = NULL,
            .will_msg = NULL,
            .will_qos = 0,
            .will_retain = 0
        };

        printf("[MQTT] Conectando...\n");
        mqtt_client_connect(mqtt_client, &broker_ip, MQTT_PORT_BROKER, mqtt_connection_cb, NULL, &ci);
    } else {
        printf("[DNS] Falha na resolução\n");
    }
}

void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
    if (status == MQTT_CONNECT_ACCEPTED) {
        printf("[MQTT] Conectado!\n");
        mqtt_connected = true;

        mqtt_set_inpub_callback(client, mqtt_incoming_publish_cb, mqtt_incoming_data_cb, NULL);

        err_t err = mqtt_subscribe(client, MQTT_TOPIC_SUB, 0, NULL, NULL);
        if (err != ERR_OK) {
            printf("[MQTT] Erro subscribe: %d\n", err);
        } else {
            printf("[MQTT] Inscrito no topico %s\n", MQTT_TOPIC_SUB);
        }

        ssd1306_clear();
        ssd1306_draw_string(0, 0, "MQTT Conectado");
        ssd1306_show();
    } else {
        printf("[MQTT] Falha conexão: %d\n", status);
        mqtt_connected = false;
    }
}
```

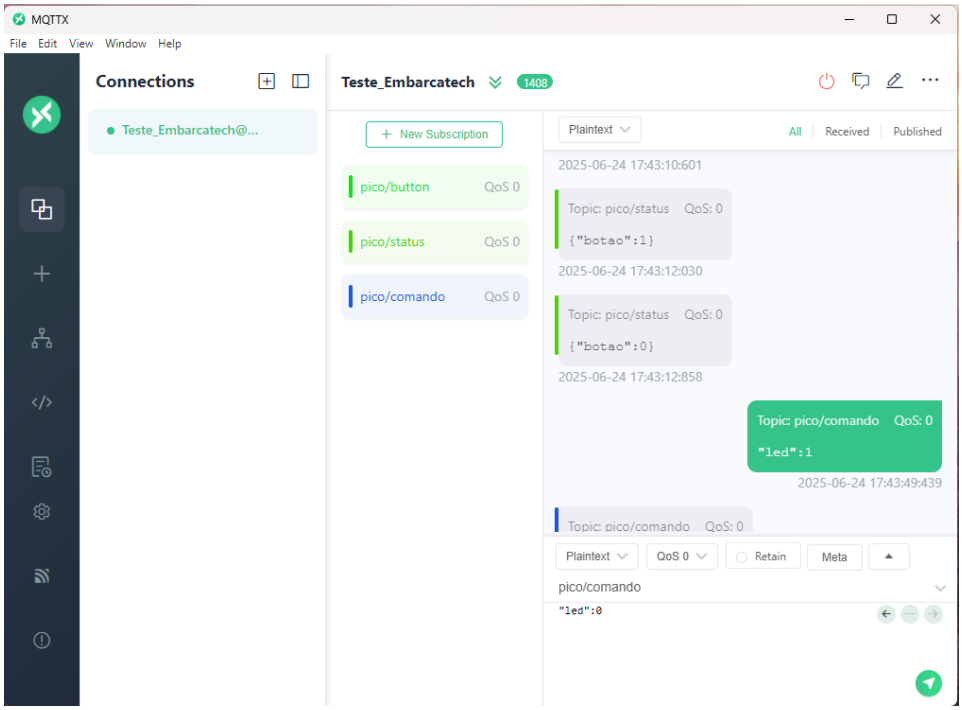
```c
void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len) {
    printf("[MQTT] Mensagem recebida no topico: %s\n", topic);
}

void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags) {
    char msg[len + 1];
    memcpy(msg, data, len);
    msg[len] = 0;
    printf("[MQTT] Payload: %s\n", msg);

    if (strstr(msg, "\"led\":1")) {
        gpio_put(LED_GPIO, 1);
    } else if (strstr(msg, "\"led\":0")) {
        gpio_put(LED_GPIO, 0);
    }

    ssd1306_clear();
    ssd1306_draw_string(0, 0, "Recebido:");
    ssd1306_draw_string(0, 10, msg);
    ssd1306_draw_string(0, 30, "LED:");
    ssd1306_draw_string(40, 30, gpio_get(LED_GPIO) ? "ON" : "OFF");
    ssd1306_show();
}

void publish_button_state(bool pressed) {
    if (!mqtt_connected) return;

    char payload[64];
    snprintf(payload, sizeof(payload), "{\"button\":%d}", pressed ? 1 : 0);

    err_t err = mqtt_publish(mqtt_client, MQTT_TOPIC_PUB, payload, strlen(payload), 0, 0, NULL, NULL);
    if (err == ERR_OK) {
        printf("[MQTT] Publicado: %s\n", payload);
    } else {
        printf("[MQTT] Erro publicar: %d\n", err);
    }
}
```

Após o desenvolvimento das funções, montamos nosso algoritmo para a função principal, conforme mostram as imagens abaixo.

```c
// ====================== Função Principal ======================
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("\n=== Iniciando ===\n");

    // Inicializa Wi-Fi
    if (cyw43_arch_init()) {
        printf("Erro na inicialização Wi-Fi\n");
        return -1;
    }
    cyw43_arch_enable_sta_mode();

    printf("[Wi-Fi] Conectando...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
        printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
        return -1;
    }
    printf("[Wi-Fi] Conectado!\n");

    // Inicializa I2C e OLED
    i2c_init(i2c1, 400000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    ssd1306_init(i2c1);
    ssd1306_clear();
    ssd1306_draw_string(0, 0, "Wi-Fi OK");
    ssd1306_draw_string(0, 10, "Conectando MQTT");
    ssd1306_show();

    // Configura Botão
    gpio_init(BUTTON_GPIO);
    gpio_set_dir(BUTTON_GPIO, GPIO_IN);
    gpio_pull_up(BUTTON_GPIO);

    // Configura LED
    gpio_init(LED_GPIO);
    gpio_set_dir(LED_GPIO, GPIO_OUT);

    // Inicializa MQTT
    mqtt_client = mqtt_client_new();
    err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_found_cb, NULL);
    if (err == ERR_OK) {
        dns_found_cb(MQTT_BROKER, &broker_ip, NULL);
    } else if (err == ERR_INPROGRESS) {
        printf("[DNS] Resolvendo...\n");
    } else {
        printf("[DNS] Erro DNS: %d\n", err);
        return -1;
    }
```

```
    // Loop Principal
    while (true) {
        cyw43_arch_poll();

        bool button_state = !gpio_get(BUTTON_GPIO);
        if (button_state != button_last_state) {
            printf("[BOTAO] %s\n", button_state ? "ON" : "OFF");
            publish_button_state(button_state);

            ssd1306_clear();
            ssd1306_draw_string(0, 0, "Botao:");
            ssd1306_draw_string(50, 0, button_state ? "ON" : "OFF");
            ssd1306_draw_string(0, 10, "LED:");
            ssd1306_draw_string(50, 10, gpio_get(LED_GPIO) ? "ON" : "OFF");
            ssd1306_show();

            button_last_state = button_state;
        }

        sleep_ms(200);
    }

    cyw43_arch_deinit();
    return 0;
}
```

Como na versão anterior, utilizamos o “MQTTX” como cliente para enviar e receber os comandos MQTT do broker da HiveMQ.



Fotos da placa BigDogLab executando o firmware:

Desafios:

Servidor em nuvem: Refaça as tarefas anteriores, utilizando outro servidor MQTT, como por exemplo: AWS, Google, MQTTX e entre outros.

Desafio 1: Utilizando a placa Bitdoglab, crie um programa para monitorar o status de um pino.

"*A Amazon Web Services (AWS), assim como outras BigTechs oferece um nível gratuito para novos usuários, permitindo o acesso a diversos serviços por um período limitado ou com uso limitado. Este nível gratuito não tem prazo de validade, mas é importante verificar os limites de uso de cada serviço, pois após ultrapassar esses limites, pode haver cobranças.*"

Depois de ver essa informação e que é obrigatório o cadastro de um cartão de crédito, decidi fazer meu teste com nuvem no broker MQTTX, que tem um acesso grátis sem o uso do cartão.

Abaixo o código utilizado, para acesso a este novo broker:

Conforme podemos perceber, não temos muitas mudanças em relação ao acesso anterior, até porque, os broker utilizados até o momento são todos em nuvem, apenas

utilizo um cliente em meu computador para fazer a inscrição nos tópicos utilizados e receber os dados.

Iniciamos com os includes necessários, definições e protótipo das funções:

DesafioMQTT1.c M

DesafioMQTT1.c > MQTT_BROKER

```c
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "pico/cyw43_arch.h"
#include "lwip/apps/mqtt.h"
#include "lwip/ip_addr.h"
#include "lwip/dns.h"

// Wi-Fi
#define WIFI_SSID "xxxxxxxxxx"
#define WIFI_PASSWORD "xxxxxxxxxx"

// MQTT Broker público compatível com MQTTX
#define MQTT_BROKER "broker.emqx.io"
#define MQTT_BROKER_PORT 1883
#define MQTT_TOPIC "embarca/status"
#define BUTTON_GPIO 5

static mqtt_client_t *mqtt_client;
static ip_addr_t broker_ip;
static bool mqtt_connected = false;
static bool dns_resolved = false;
```

```c
// Funções
static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
void publish_msg(bool button_pressed, float temp_c);
float read_temperature();
void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg);
```

Nosso *main*:

```c
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("== Publicador MQTT - Pico W + MQTTX ==\n");

    if (cyw43_arch_init()) {
        printf("Erro no Wi-Fi\n");
        return -1;
    }
    cyw43_arch_enable_sta_mode();

    printf("[Wi-Fi] Conectando...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
        printf("[Wi-Fi] Falha\n");
        return -1;
    }
    printf("[Wi-Fi] Conectado!\n");

    gpio_init(BUTTON_GPIO);
    gpio_set_dir(BUTTON_GPIO, GPIO_IN);
    gpio_pull_up(BUTTON_GPIO);

    adc_init();
    adc_set_temp_sensor_enabled(true);
    adc_select_input(4);

    mqtt_client = mqtt_client_new();
    if (mqtt_client == NULL) {
        printf("[MQTT] Erro ao criar cliente\n");
        return -1;
    }
```

```
    err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_check_callback, NULL);
    if (err == ERR_OK) {
        dns_check_callback(MQTT_BROKER, &broker_ip, NULL);
    } else if (err == ERR_INPROGRESS) {
        printf("[DNS] Resolvendo %s...\n", MQTT_BROKER);
    } else {
        printf("[DNS] Erro: %d\n", err);
        return -1;
    }

    while (!dns_resolved || !mqtt_connected) {
        cyw43_arch_poll();
        sleep_ms(10);
    }

    while (true) {
        cyw43_arch_poll();

        bool button_state = !gpio_get(BUTTON_GPIO);
        float temp = read_temperature();

        publish_msg(button_state, temp);

        sleep_ms(1000);
    }

    cyw43_arch_deinit();
    return 0;
}
```

Nossas funções:

```
void publish_msg(bool button_pressed, float temp_c) {
    if (!mqtt_connected) return;

    char payload[128];
    snprintf(payload, sizeof(payload),
             "{\"botao\":\"%s\",\"temperatura\":%.2f}",
             button_pressed ? "ON" : "OFF", temp_c);

    mqtt_publish(mqtt_client, MQTT_TOPIC, payload, strlen(payload), 0, 0, NULL, NULL);
    printf("[MQTT] Enviado: %s\n", payload);
}

float read_temperature() {
    const int samples = 5;
    uint32_t total = 0;
    for (int i = 0; i < samples; i++) {
        total += adc_read();
        sleep_ms(2);
    }

    float avg = total / (float)samples;
    const float factor = 3.3f / (1 << 12);
    float voltage = avg * factor;
    float temp = 27.0f - (voltage - 0.706f) / 0.001721f;
    return temp;
}

void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
    if (status == MQTT_CONNECT_ACCEPTED) {
        printf("[MQTT] Conectado ao broker!\n");
        mqtt_connected = true;
    } else {
        printf("[MQTT] Falha (%d)\n", status);
        mqtt_connected = false;
    }
}
```

```c
void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
    if (ipaddr) {
        broker_ip = *ipaddr;
        dns_resolved = true;
        printf("[DNS] %s -> %s\n", name, ipaddr_ntoa(ipaddr));

        struct mqtt_connect_client_info_t ci = {
            .client_id = "pico-w-client",
            .keep_alive = 60,
            .client_user = NULL,
            .client_pass = NULL,
            .will_topic = NULL,
            .will_msg = NULL,
            .will_qos = 0,
            .will_retain = 0
        };

        mqtt_client_connect(mqtt_client, &broker_ip, MQTT_BROKER_PORT, mqtt_connection_callback, NULL, &ci);
    } else {
        printf("[DNS] Falha para %s\n", name);
    }
}
```

Desafio 2: Reescrever o programa 2 com outro broker em nuvem:

src > DesafioMQTT2.c > WIFI_PASSWORD

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include "pico/cyw43_arch.h"
#include "lwip/apps/mqtt.h"
#include "lwip/ip_addr.h"
#include "lwip/dns.h"

#include "ssd1306.h"

// ======================== Configurações ========================
#define WIFI_SSID "xxxxxxxxx"
#define WIFI_PASSWORD "xxxxxxxxx"

#define MQTT_BROKER "broker.emqx.io"        // Broker da MQTTX
#define MQTT_PORT_BROKER 1883               // Porta sem TLS

#define MQTT_TOPIC_PUB "embarca/status"
#define MQTT_TOPIC_SUB "embarca/comando"

#define BUTTON_GPIO 5
#define LED_GPIO 12

#define I2C_SDA 14
#define I2C_SCL 15

// ======================== Variáveis Globais ====================
static mqtt_client_t *mqtt_client;
static ip_addr_t broker_ip;
static bool mqtt_connected = false;
static bool button_last_state = false;
```


```c
// ======================== Variáveis Globais ====================
static mqtt_client_t *mqtt_client;
static ip_addr_t broker_ip;
static bool mqtt_connected = false;
static bool button_last_state = false;
```

Nossos protótipos de funções:

```c
// ====================== Protótipos ==========================
void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len);
void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags);
void publish_button_state(bool pressed);
void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg);
```

Nosso “*main*”:

```c
// ====================== Função Principal ======================
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("\n=== Iniciando ===\n");

    // Inicializa Wi-Fi
    if (cyw43_arch_init()) {
        printf("Erro na inicialização Wi-Fi\n");
        return -1;
    }
    cyw43_arch_enable_sta_mode();

    printf("[Wi-Fi] Conectando...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
        printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
        return -1;
    }
    printf("[Wi-Fi] Conectado!\n");

    // Inicializa I2C e OLED
    i2c_init(i2c1, 400000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    ssd1306_init(i2c1);
    ssd1306_clear();
    ssd1306_draw_string(0, 0, "Wi-Fi OK");
    ssd1306_draw_string(0, 10, "Conectando MQTT");
    ssd1306_show();

    // Configura Botão
    gpio_init(BUTTON_GPIO);
    gpio_set_dir(BUTTON_GPIO, GPIO_IN);
    gpio_pull_up(BUTTON_GPIO);

    // Configura LED
    gpio_init(LED_GPIO);
    gpio_set_dir(LED_GPIO, GPIO_OUT);
```

```c
    // Inicializa MQTT
    mqtt_client = mqtt_client_new();
    err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_found_cb, NULL);
    if (err == ERR_OK) {
        dns_found_cb(MQTT_BROKER, &broker_ip, NULL);
    } else if (err == ERR_INPROGRESS) {
        printf("[DNS] Resolvendo...\n");
    } else {
        printf("[DNS] Erro DNS: %d\n", err);
        return -1;
    }

    // Loop Principal
    while (true) {
        cyw43_arch_poll();

        bool button_state = !gpio_get(BUTTON_GPIO);
        if (button_state != button_last_state) {
            printf("[BOTAO] %s\n", button_state ? "ON" : "OFF");
            publish_button_state(button_state);

            ssd1306_clear();
            ssd1306_draw_string(0, 0, "Botao:");
            ssd1306_draw_string(50, 0, button_state ? "ON" : "OFF");
            ssd1306_draw_string(0, 10, "LED:");
            ssd1306_draw_string(50, 10, gpio_get(LED_GPIO) ? "ON" : "OFF");
            ssd1306_show();

            button_last_state = button_state;
        }

        sleep_ms(200);
    }

    cyw43_arch_deinit();
    return 0;
}
```

Nossas funções:

```c
// ================= Callbacks e Publicação ====================
void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
    if (ipaddr != NULL) {
        broker_ip = *ipaddr;
        printf("[DNS] %s -> %s\n", name, ipaddr_ntoa(ipaddr));

        struct mqtt_connect_client_info_t ci = {
            .client_id = "pico-client",
            .keep_alive = 60,
            .client_user = NULL,
            .client_pass = NULL,
            .will_topic = NULL,
            .will_msg = NULL,
            .will_qos = 0,
            .will_retain = 0
        };

        printf("[MQTT] Conectando...\n");
        mqtt_client_connect(mqtt_client, &broker_ip, MQTT_PORT_BROKER, mqtt_connection_cb, NULL, &ci);
    } else {
        printf("[DNS] Falha na resolução\n");
    }
}
```

```c
void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
    if (status == MQTT_CONNECT_ACCEPTED) {
        printf("[MQTT] Conectado!\n");
        mqtt_connected = true;

        mqtt_set_inpub_callback(client, mqtt_incoming_publish_cb, mqtt_incoming_data_cb, NULL);

        err_t err = mqtt_subscribe(client, MQTT_TOPIC_SUB, 0, NULL, NULL);
        if (err != ERR_OK) {
            printf("[MQTT] Erro subscribe: %d\n", err);
        } else {
            printf("[MQTT] Inscrito no topico %s\n", MQTT_TOPIC_SUB);
        }

        ssd1306_clear();
        ssd1306_draw_string(0, 0, "MQTT Conectado");
        ssd1306_show();
    } else {
        printf("[MQTT] Falha conexão: %d\n", status);
        mqtt_connected = false;
    }
}

void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len) {
    printf("[MQTT] Msg no tópico: %s\n", topic);
}
```

```c
void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags) {
    char msg[len + 1];
    memcpy(msg, data, len);
    msg[len] = 0;
    printf("[MQTT] Payload: %s\n", msg);

    if (strstr(msg, "\"led\":\"ON\"")) {
        gpio_put(LED_GPIO, 1);
    } else if (strstr(msg, "\"led\":\"OFF\"")) {
        gpio_put(LED_GPIO, 0);
    }

    ssd1306_clear();
    ssd1306_draw_string(0, 0, "Recebido:");
    ssd1306_draw_string(0, 10, msg);
    ssd1306_draw_string(0, 30, "LED:");
    ssd1306_draw_string(40, 30, gpio_get(LED_GPIO) ? "ON" : "OFF");
    ssd1306_show();
}

void publish_button_state(bool pressed) {
    if (!mqtt_connected) return;

    char payload[64];
    snprintf(payload, sizeof(payload), "{\"botao\":\"%s\"}", pressed ? "ON" : "OFF");

    err_t err = mqtt_publish(mqtt_client, MQTT_TOPIC_PUB, payload, strlen(payload), 0, 0, NULL, NULL);
    if (err == ERR_OK) {
        printf("[MQTT] Publicado: %s\n", payload);
    } else {
        printf("[MQTT] Erro publicar: %d\n", err);
    }
}
```

Todos os nossos códigos trabalham em nuvem, por isso utilizamos a função para acesso a um servidor DNS para localizar o IP fixo do broker na nuvem.

Os teste são similares as versões iniciais.