Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. Antonio selvatici

Placa ESP8266 NodeMCU

O ESP8266 NodeMCU é uma placa de desenvolvimento contendo o módulo ESP8266-12E da Espressif, um microprocessador com rádio WiFi que pode ser programado de várias maneiras. Para nós, o modo mais simples de programá-lo é através da IDE Arduíno. Aqui vamos programá-lo para capturar as informações dos sensores e controlar nossos atuadores, enviando a um servidor via webservice, MQTT ou outro protocolo de mensagens. O código não irá mudar muito do que vimos anteriormente para o Arduíno, com exceção de que a lógica do WiFi será aqui incluída.

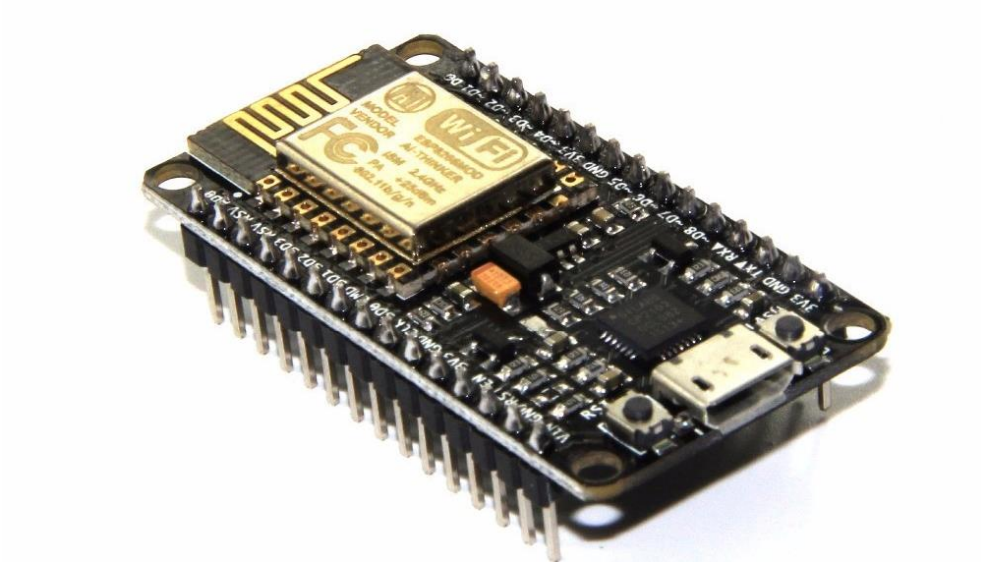


Figura 1: ESP8266 NodeMCU – vista em perspectiva

Abaixo você pode ver as principais características do módulo ESP8266-12E:

– Arquitetura RISC de 32 bits
– Pode operar em 80MHz / 160MHz
– 4Mb de memória flash
– 64Kb de RAM para instruções
– 96Kb de RAM para dados
– WiFi nativo padrão 802.11b/g/n
– Opera em modo AP, Station ou AP + Station
– Pinos digitais, exceto o D0 possuem interrupção, PWM, I2C e one wire
– Pinos operam em nível lógico de 3.3V

O ESP 8266 possui funciona com lógica de 3,3V, ou seja, suas as entradas e saídas **não devem** ser usadas diretamente para enviar ou receber dados de um Arduino UNO (5V) sem realizar uma conversão de tensão.

Veja que este microcontrolador é muito mais poderoso que o do Ardunio, operando de 5 a 10 vezes mais rápido e com mais memória de armazenamento Flash, que pode ser usado para guardar programas mais complexos ou mesmo ser usado como memória de armazenamento não volátil para aplicações.

A placa de desenvolvimento

Esta plataforma é composta basicamente por um módulo ESP8266-12E, uma porta micro USB para alimentação e programação, conversor USB serial integrado e já possui WiFi nativo.

Como diferenças para o Arduino Uno, podemos notar que a placa da desenvolvimento Node MCU exporta nove portas digitais de propósito geral (D0 a D8), e apenas uma porta com capacidade de leitura analógica (A0). Todas as portas digitais, com exeção de D0, possuem interrupção, PWM, e suportam comunicação com os padrões I2C e one wire. Outro diferencial do NodeMCU é a possibilidade de fazer a programação da placa via OTA (Over The Air), ou seja, através do WiFi você pode enviar os códigos para a placa.

O módulo ESP8266 tem suporte de fábrica para programação em linguagem Lua, pois o firmware instalado já vem com um interpretador. No entanto, podemos usar a IDE do Arduino para fazer a programação, com a diferença que todo o firmware (conteúdo da memória Flash) é sobrescrito, apagando o firmware original.

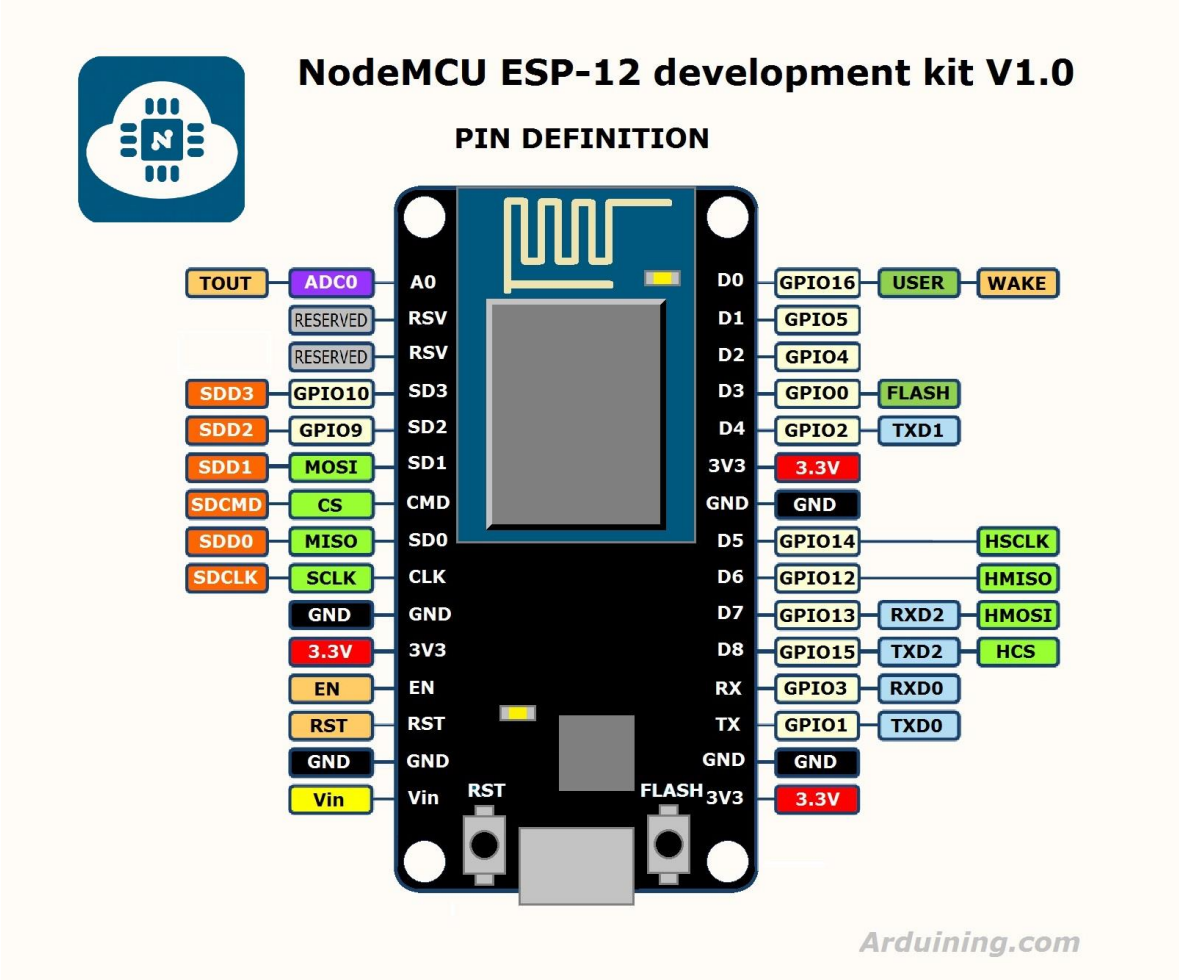


Figura 2: ESP8266 NodeMCU – vista superior e detalhes das portas

Instalando o suporte à placa na IDE do Arduíno

Para programar o ESP8266 NodeMCU é preciso inicialmente instalar o suporte à programação desta placa na IDE do Arduíno. Esse processo requer uma versão bem atual da IDE, a partir da 1.6.4, pois as versões mais novas possuem um gerenciador de placas suportadas pela IDE. Um breve tutorial sobre como realizar essa configuração automaticamente encontra-se em: http://josecintra.com/blog/programando-nodemcu-esp8266-ide-arduino/:

1. Abra a janela de Preferências e digite no campo “Additional Board Manager URLs” o seguinte endereço: http://arduino.esp8266.com/stable/package_esp8266com_index.json

2. Instalar o pacote do ESP8266 atravpes do gerenciador de placas: Tools -> Board Manager... e então procurar por ESP8266 (instalar a última versão)

3. No menu Tools, configure a sua placa de acordo com o modelo que está usando. As opções mais comuns são:
    a. Board: NodeMCU, de acordo com seu modelo;
    b. CPU Frequency: 80 MHz;
    c. Upload Speed: 115200

Após o script baixar e instalar todo o software necessário, abra a IDE do Arduíno e selecione a placa NodeMCU 1.0 para desenvolvimento, como mostra a figura abaixo:

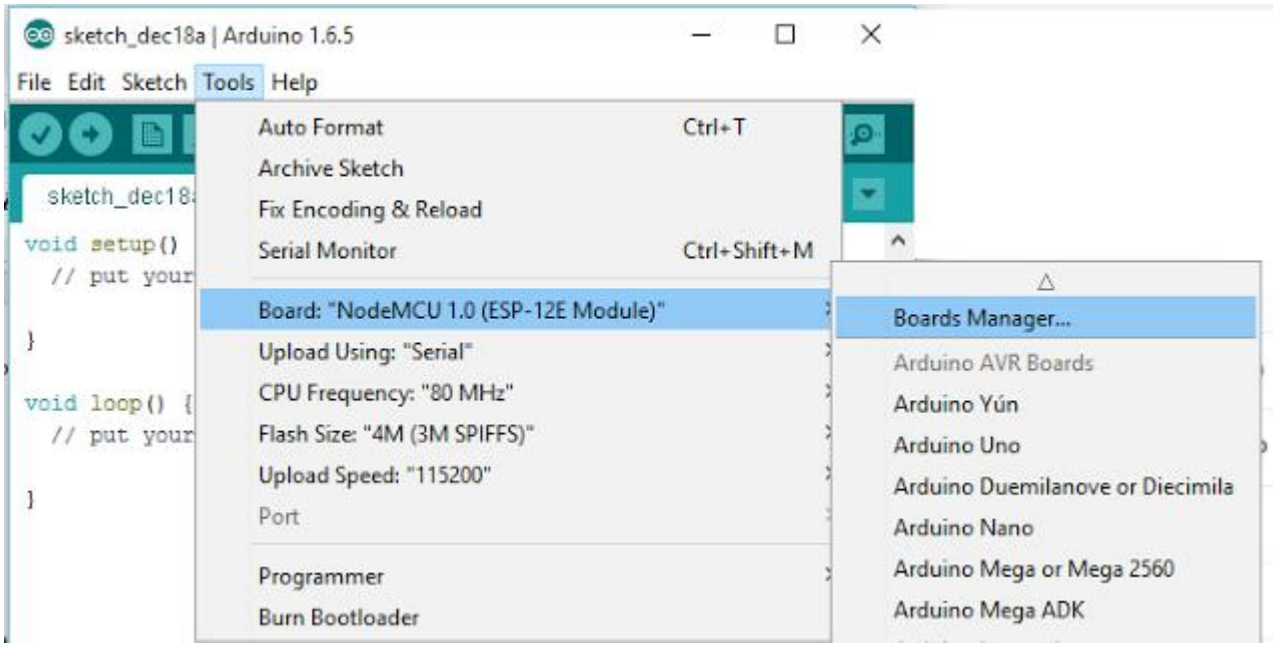


Figura 3: Selecionando a placa correta para o desenvolvimento

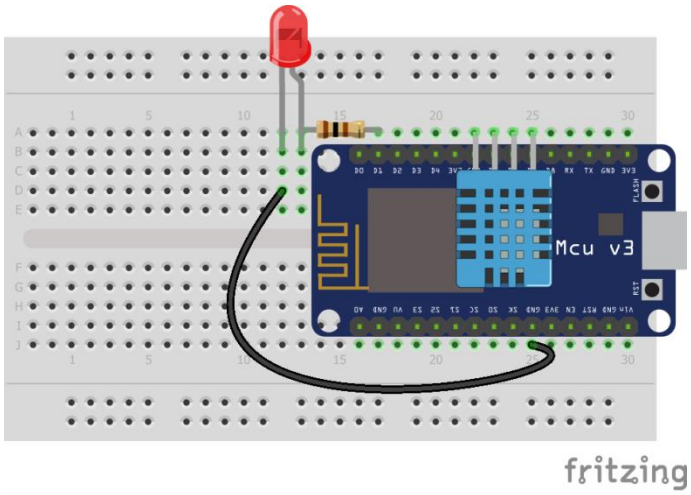
Por fim, precisamos instalar o driver CH341 para que o Windows reconheça a placa como uma porta serial do sistema. O driver pode ser baixado do site http://www.electrodragon.com/w/CH341, ou, por comodidade, pode ser baixado na pasta compartilhada no Google Drive.

Características de programação da placa ESP8266-12E

Programar o ESP8266 NodeMCU é praticamente igual a programar uma placa Arduíno, sendo que a principal difireneça é a disposição das portas:

- O ESP8266-12E possui apenas uma porta analógica (ADC). Portanto, só podemos usar a função `analogRead()` com a porta `A0`
- O ESP8266-12 possui apenas nove portas digitais, acessíveis através das constantes `D0`, `D1`, … `D8`, sendo que apenas a porta D0 não é PWM. Tentar acessar a pinagem do ESP8266-12E diretamente através de seu identificador numérico poderá levar a erros de endereçamento.

O circuito a ser usado para o lab de hoje está apresentado abaixo:



O HTTPClient

Para usar as funções de WiFi do ESP8266 NodeMCU, importamos o cabeçalho “ESP8266WiFi.h”, que exporta o objeto `WiFi`, e o cabeçalho “ESP8266HTTPClient.h”, que exporta a classe HTTPClient. O primeiro permite gerenciar a conexão do dispositivo com um Access Point (AP), e a segunda permite usar um cliente HTTP usando o WiFi do ESP8266.

- Para conectar o ESP8266 a um Access Point com nome SSID e senha de acesso PASSWORD, fazemos: `WiFi.begin("SSID", "PASSWORD");`

- Para verificar se o WiFi está conectado ao AccessPoint, fazemos a comparação:
  `WiFi.status() != WL_CONNECTED`
- Para enviar o IP obtido via DHCP pela porta serial, fazemos:
  `Serial.println(WiFi.localIP());`

Já a classe HTTPClient permite enviar requisições HTTP via WiFi.

- Para iniciar uma requisição POST em um servidor com endereço “example.com” na porta 80, fazemos:
  ```
  HTTPClient http;
  http.begin("example.com:80");
  ```
- Para enviar dados no formato JSON na variável JSONString pela conexão criada, fazemos:
  ```
  http.addHeader("Content-Type","application/json");
  int statusCode = http.POST(JSONString);
  ```
- Para ler a resposta do servidor, fazemos:
  ```
  String payload = http.getString();
  ```

O programa abaixo envia os dados dos sensores do DHT11 ao servidor no endereço http://centralhub.mybluemix.net/nodemcu_http_sensors, enquanto controla a luz do LED na porta D1 a partir do comando recebido em http://centralhub.mybluemix.net/nodemcu_http_led. Para interagir com as placas que estão consumindo o serviço, basta usar o navegador para acessar o endereço: http://centralhub.mybluemix.net/nodemcu_http

Segue o programa:

```
/*
 *  Este programa envia dados do DHT a uma API HTTP
 *  Também lê os comandos para o LED vindos da API
 */

#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>
#include <DHT.h>

#define ID 100
#define LED D1
//Aqui nós alimentamos o DHT com as portas digitais mesmo
#define DHTVCC D7
#define DHTPIN D6

#define SSID "FIAP-WIFI"
```

```
#define PASSWD ""

#define HOST "centralhub.mybluemix.net"
#define PORT 80
#define SENSORS_URL "/nodemcu_http_sensors"
#define LED_URL "/nodemcu_http_led"
#define TIMEOUT 5000 //milliseconds - conection timeout
#define REFRESH 1000 //milliseconds - refresh rate for the led and sensors
data

//Variáveis globais
boolean ledon = false;
DHT dht(DHTPIN,DHT11);

//Configuracao da placa
void setup() {
  //Led
  pinMode(LED,OUTPUT);
  //DHT
  pinMode(DHTVCC, OUTPUT);
  digitalWrite(DHTVCC,HIGH); //liga o DHT
  dht.begin(); //Inicializa o DHT

  //Conexão serial
  Serial.begin(9600);

  //Conexão wifi
  Serial.println("Inicia o processo de conexão ao AP");
  WiFi.begin(SSID, PASSWD);
}

//Codigo do programa principal
void loop() {
  /*** PARTE 1 - Saida de dados ***/
  //Leitura dos sensores analogicos
  float temp = dht.readTemperature();
  float umid = dht.readHumidity();

  //Tenta enviar o dado pelo WiFi
  Serial.println("Checa conexão ao AP");
  if(WiFi.status() == WL_CONNECTED) {
      Serial.println("Conectado ao AP");
      HTTPClient http;
      //Aqui setamos o endereço e a porta do serviço
      http.begin(HOST,PORT,SENSORS_URL);
```

```
      //Cria a string JSON em buffer
      StaticJsonBuffer<200> jsonBuffer;
      JsonObject& json = jsonBuffer.createObject();
      json["id"] = ID;
      json["ip"] = String(WiFi.localIP());
      if(!isnan(temp)) json["temp"] = temp;
      if(!isnan(umid)) json["umid"] = umid;

      //Criando a string a ser enviada via http post
      char buffer[256];
      json.printTo(buffer,sizeof(buffer));
      //Envia buffer pelo cliente http via POST
      Serial.print("[HTTP] POST...\n");
      // start connection and send HTTP header
      int httpCode = http.POST(String(buffer));
      // httpCode will be negative on error
      if(httpCode > 0) {
        // HTTP header has been send and Server response header has been
handled
        Serial.printf("[HTTP] POST... code: %d\n", httpCode);
        Serial.printf("[HTTP] POST... response: %s\n",
http.getString().c_str());
      } else {
        Serial.printf("[HTTP] POST... failed, error: %s\n",
http.errorToString(httpCode).c_str());
      }

      /*** PARTE 2 - Entrada de dados ***/
      // Busca no servidor pelo comando de acender o LED
      http.end();
      http.begin(HOST,PORT,LED_URL);
      //Envia pedido pelo cliente http via GET
      Serial.print("[HTTP] GET...\n");
      // start connection and send HTTP header
      httpCode = http.GET();
      // httpCode will be negative on error
      String payload;
      if(httpCode > 0) {
        // HTTP header has been send and Server response header has been
handled
        payload = http.getString();
        Serial.printf("[HTTP] GET... code: %d\n", httpCode);
        Serial.printf("[HTTP] GET... response: %s\n", payload.c_str());
      } else {
```

```
          Serial.printf("[HTTP] GET... failed, error: %s\n",
http.errorToString(httpCode).c_str());
      }

      if (httpCode == 200) {
        StaticJsonBuffer<200> jsonBuffer;
        JsonObject& json = jsonBuffer.parseObject(payload.c_str());
        if(json.success() && json.containsKey("value")) {
          analogWrite(LED, json["value"]);
        }
        delay(10);
      } //if httCode == 200
  }//if wifi connected
  else
  {
    Serial.println("Não conectado ao AP");
  }
  delay(REFRESH);
}
```

O cliente MQTT

Para utilizar o NodeMCU (e as placas Arduino em geral) como clientes MQTT, podemos usar a biblioteca `PubSubClient`, instalável via gerenciador de bibliotecas (pode ser a versão mais recente). Ela usa o cliente de WiFi como meio de transporte das mensagens.

Utilizar a biblioteca é simples. Primeiramente devemos instanciar o objeto cliente PubSub, informando o cliente de WiFi, o endereço do servidor e a função de call-back chamada sempre que chegar uma mensagem.:

```
char mqtt_server[] = "iot.eclipse.org";
WiFiClient espClient;
PubSubClient client(espClient);
client.setServer(mqtt_server, 1883);
client.setCallback(callback);
```

Então podemos tentar a conexão com o servidor, informando um CLientID, que pode ser gerado randomicamente. É aconselhável que haja uma verificação periódica do status da conexão, e, caso ela tenha se encerrado, seja restabelecida.

```
String clientId = "ESP8266Client-";
clientId += String(random(0xffff), HEX);
bool status = client.connect(clientId.c_str()))
```

Logo após a conexão, devemos subscrever os tópicos desejados através de `cliente.subscribe(topic)`:

```
if (status) {
      client.subscribe("inTopic");
}
```

Por fim, para publicar uma mensagem, usamos o método `client.publish(topic, message)`:

```
client.publish("outTopic", "Hello World!!!");
```

O programa abaixo é muito semelhante ao programa anterior, porém usa o cliente MQTT em vez do cliente HTTP. Ele envia os dados dos sensores do DHT11 ao servidor iot.eclipse.org no tópico `fiap/iot/devtype/nodemcu/id/100/sensors`, enquanto controla a luz do LED na porta D1 a partir do comando recebido em `fiap/iot/devtype/nodemcu/id/100/cmd/led`. Para interagir com as placas que estão consumindo o serviço, basta usar o navegador para acessar o endereço: http://centralhub.mybluemix.net/nodemcu_mqtt. Você pode modificar o ID do dispositivo dentro do código e do tópico MQTT para melhor controla-lo.

```
/*
    Este programa envia dados do DHT a uma um tópico MQTT
    Também lê os comandos para o LED a partir de mensagens subscritas
*/

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DHT.h>

#define ID 100
#define LED D1
//Aqui nós alimentamos o DHT com as portas digitais mesmo
#define DHTVCC D7
#define DHTPIN D6

#define SSID "FIAP-WIFI"
#define PASSWD ""

#define HOST "iot.eclipse.org"
#define PORT 1883
#define SENSORS_TOPIC "fiap/iot/devtype/nodemcu/id/100/sensors"
#define LED_TOPIC "fiap/iot/devtype/nodemcu/id/100/cmd/led"
```

```
#define TIMEOUT 5000 //milliseconds - conection timeout
#define REFRESH 1000 //milliseconds - refresh rate for the led and sensors
data

//Variáveis globais
boolean ledon = false;
DHT dht(DHTPIN, DHT11);
WiFiClient espClient;
PubSubClient mqtt(espClient);
long lastMsg;

//Função de callback que deve ser executada quando chega uma mensagem
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print(String("Message arrived [") + topic + "] ");
  payload[length] = '\0'; //com sorte há espaço para mais um caractere...
  StaticJsonBuffer<200> jsonBuffer;
  JsonObject& json = jsonBuffer.parseObject(payload);
  if (json.success() && json.containsKey("value")) {
    analogWrite(LED, json["value"]);
  }
  delay(10);
}

//Função que verifica a conexão do cliente MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!mqtt.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (mqtt.connect(clientId.c_str())) {
      Serial.println("MQTT client connected");
      // Once connected, resubscribe to LED topic
      mqtt.subscribe(LED_TOPIC);
    } else {
      Serial.print("failed, rc=");
      Serial.print(mqtt.state());
      Serial.println(String(" try again in ") + TIMEOUT + "
milliseconds");
      // Wait TIMEOUT seconds before retrying
      delay(TIMEOUT);
    }
  }
```

```
}

//Configuracao da placa
void setup() {
  lastMsg = millis();
  //Led
  pinMode(LED, OUTPUT);
  //DHT
  pinMode(DHTVCC, OUTPUT);
  digitalWrite(DHTVCC, HIGH); //liga o DHT
  dht.begin(); //Inicializa o DHT
  //Conexão serial
  Serial.begin(9600);
  //Conexão wifi
  Serial.println("Inicia o processo de conexão ao AP");
  WiFi.begin(SSID, PASSWD);
  //Configuração MQTT
  mqtt.setServer(HOST, PORT);
  mqtt.setCallback(callback);
}

//Codigo do programa principal
void loop() {
  /*** PARTE 1 - Saida de dados ***/
  //Leitura dos sensores analogicos
  float temp = dht.readTemperature();
  float umid = dht.readHumidity();

  //Tenta enviar o dado pelo WiFi
  if (WiFi.status() == WL_CONNECTED) {
    //Verifica a conexão com o broker MQTT
    // e a renova se for o caso
    reconnect();
    //Checa mensagens - deve ser chamado de forma recorrente
    mqtt.loop();
    long now = millis();
    if (now - lastMsg > REFRESH) {
      //Cria a string JSON em buffer
      StaticJsonBuffer<200> jsonBuffer;
      JsonObject& json = jsonBuffer.createObject();
      json["id"] = ID;
      json["ip"] = String(WiFi.localIP());
      if (!isnan(temp)) json["temp"] = temp;
      if (!isnan(umid)) json["umid"] = umid;
```

```
      //Criando a string a ser enviada via MQTT publish
      char buffer[256];
      json.printTo(buffer, sizeof(buffer));
      //Envia buffer pelo cliente http via POST
      Serial.print("[MQTT] Publish...\n");
      mqtt.publish(SENSORS_TOPIC, buffer);
      lastMsg = now;
    }
    //A parte de recepção de dados foi migrada para o callback
  }//if wifi connected
  else
  {
    Serial.println("Não conectado ao AP");
    delay(REFRESH);
  }
}
```

Referências:

http://blogmasterwalkershop.com.br/embarcados/nodemcu/nodemcu-uma-plataforma-com-caracteristicas-singulares-para-o-seu-projeto-iot/

http://blog.eletrogate.com/nodemcu-esp12-introducao-1/

https://www.filipeflop.com/blog/esp8266-nodemcu-como-programar/

https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/

https://tttapa.github.io/ESP8266/Chap04%20-%20Microcontroller.html