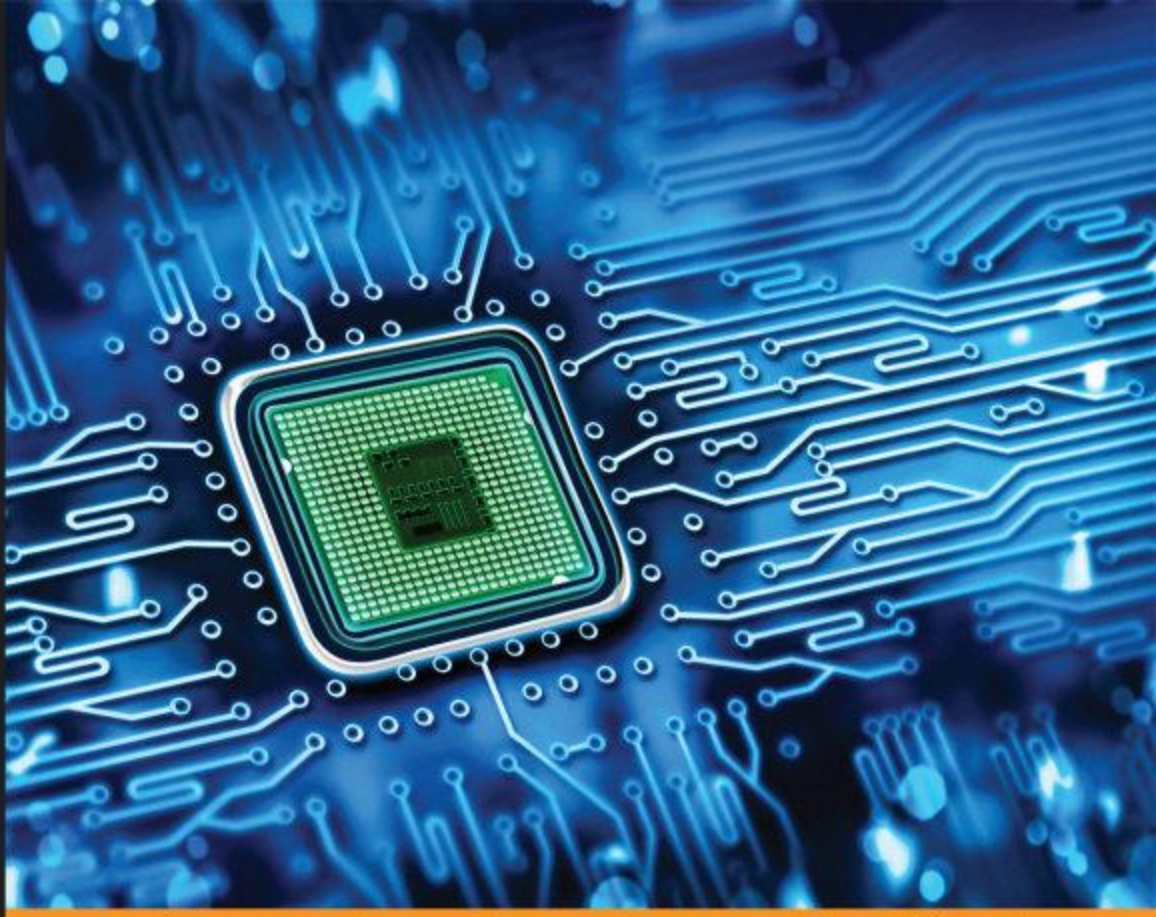


Community Experience Distilled

Интернет вещей с ESP8266

Создавайте потрясающие проекты Интернета вещей с помощью микросхемы Wi-Fi ESP8266

Марко Шварц

Интернет вещей с ESP8266

Создавайте потрясающие проекты Интернета вещей с помощью микросхемы Wi-Fi ESP8266

Марко Шварц

Об авторе

Марко Шварц - инженер-электрик, предприниматель и блогер. Он имеет степень магистра электротехники и информатики в Supélec, Франция, и степень магистра в области микроэлектроники в Ecole Polytechnique Fédérale de Lausanne (EPFL) в Швейцарии.

Он имеет более чем пятилетний опыт работы в области электротехники. Интересы Марко связаны с электроникой, домашней автоматикой, платформами Arduino и Raspberry Pi, проектами оборудования с открытым исходным кодом и 3D-печатью.

У него есть несколько веб-сайтов об Arduino, в том числе веб-сайт Open Home Automation, посвященный созданию систем домашней автоматики с использованием устройств с открытым исходным кодом.

Марко написал еще одну книгу по домашней автоматике и Arduino под названием «Домашняя автоматика с Arduino: автоматизируйте свой дом с помощью устройств с открытым исходным кодом». Он также написал книгу о том, как создавать проекты Интернета вещей с помощью Arduino, под названием «Интернет вещей с Arduino Yun».

Оглавление

Предисловие

Интернет вещей (IoT) - это захватывающая область, в которой предполагается, что все устройства, которые нас окружают, будут подключены к Интернету и взаимодействовать с нами, а также между собой. По оценкам, к 2020 году в мире будет 50 миллиардов устройств Интернета вещей.

С другой стороны, чип ESP8266 - это небольшой, дешевый (менее 5 долларов) и мощный чип Wi-Fi, который также очень легко запрограммировать. Следовательно, это просто идеальный инструмент для создания недорогих и интересных проектов Интернета вещей. В этой книге вы узнаете все, что вам нужно знать о том, как создавать проекты IoT с использованием чипа ESP8266 Wi-Fi.

О чем эта книга

Глава *1*,«Приступая к работе с ESP8266», вы научитесь как выбрать плату ESP8266 и загрузить свой первый скетч в чип.

Глава 2,«Первые проекты с ESP8266», объяснит основы ESP8266, выполнивнесколько действительно простых проектов.

В главе 3 «Облачная регистрация данных с помощью ESP8266» мы погрузимся в суть темы книги и создадим проект, который может регистрировать данные измерений в облаке.

Глава 4 «Управление устройствами из любого места» расскажет, как управлять устройствами из любой точки мира с помощью ESP8266.

Глава 5 «Взаимодействие с веб-службами» покажет, как использовать ESP8266 для взаимодействия с существующими веб-платформами, такими как Twitter.

В главе 6 «Межмашинное взаимодействие» объясняется, как заставить микросхемы ESP8266 взаимодействовать друг с другом через облако, чтобы создавать приложения, не требующие вмешательства человека.

Глава *7*, «Отправка уведомлений с ESP8266», покажет, как отправлять автоматические уведомления с ESP8266, например, с помощью текстового сообщения или электронной почты.

Глава 8 «Управление дверным замком из облака» будет создано наше первое приложение: дверной замок, которым можно управлять удаленно.

В главе 9 «Создание физического биткойн-тикера» ESP8266 будет использоваться для интересного проекта: физического отображения текущей цены биткойнов.

В главе 10 «Беспроводное садоводство с ESP8266» мы углубимся в более сложную задачу, узнав, как автоматизировать свой сад с помощью ESP8266.

Глава 11, «Облачная система домашней автоматики», покажет, как построитьосновные блоки системы домашней автоматики с помощью ESP8266.

В главе 12,«Робот ESP8266 с облачным управлением», объясняется, как использовать ESP8266 для управления мобильным роботом из любой точки мира.

Глава 13 «Создание собственной облачной платформы для управления устройствами ESP8266» расскажет, как создать нашу собственную облачную платформу для ваших проектов ESP8266.

Что вам понадобится для этой книги

Для этой книги вам понадобится IDE Arduino, которую мы будем использовать для всех проектов книги. Вы узнаете, как его установить и настроить в первой главе книги.

Главы книги также были написаны с прогрессивной сложностью, поэтому, даже если вы мало знаете об Arduino и / или ESP8266, вы сможете учиться по мере продвижения по главам. Однако предыдущий опыт программирования (особенно в C ++ и / или JavaScript) рекомендуется для этой книги.

Для кого эта книга

Эта книга предназначена для тех, кто хочет создавать мощные и недорогие проекты IoT с использованием чипа ESP8266 Wi-Fi, в том числе для тех, кто плохо знаком с IoT, или тех, кто уже имеет опыт работы с другими платформами, такими как Arduino.

1

Начало работы с ESP8266

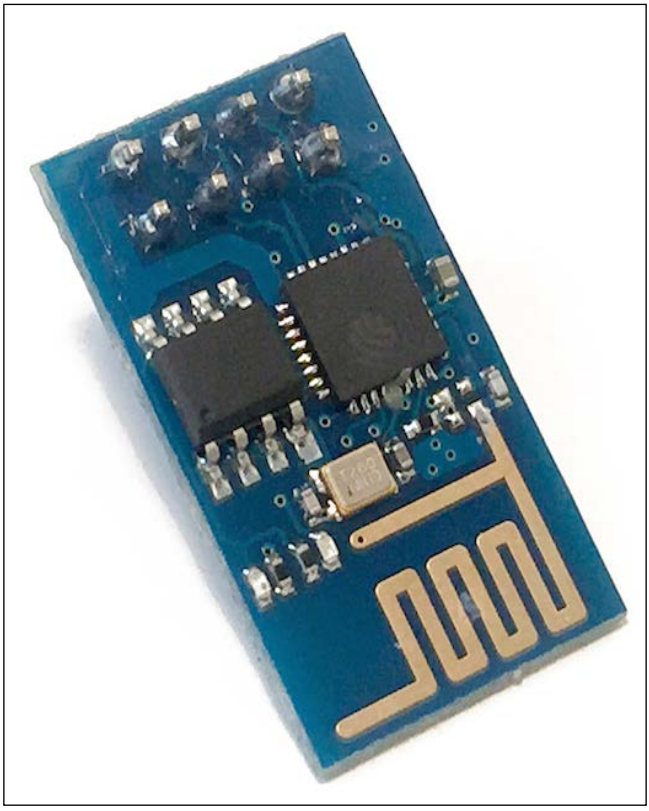
В этой главе мы начнем с настройки микросхемы ESP8266. Мы узнаем, как правильно выбрать модуль для вашего проекта и получить все дополнительное оборудование, необходимое для использования чипа. Мы также увидим, как подключить ESP8266 к вашему компьютеру, чтобы вы могли программировать его с помощью кабеля USB.

Затем мы увидим, как настроить и загрузить код в чип ESP8266. Для этого мы будем использовать IDE Arduino. Это значительно упрощает использование ESP8266, поскольку мы будем использовать хорошо известный интерфейс и язык для настройки чипа. Мы также сможем использовать большинство уже существующих библиотек Arduino для наших проектов. Давайте начнем!

Как выбрать свой модуль ESP8266

Сначала мы посмотрим, как выбрать правильный модуль ESP8266 для вашего проекта.
На рынке доступно множество модулей, и довольно легко потеряться со всеми доступными вариантами.

Первое, о чем вы, наверное, слышали, - это небольшой последовательный беспроводной порт ESP8266.
Модуль приемопередатчика:



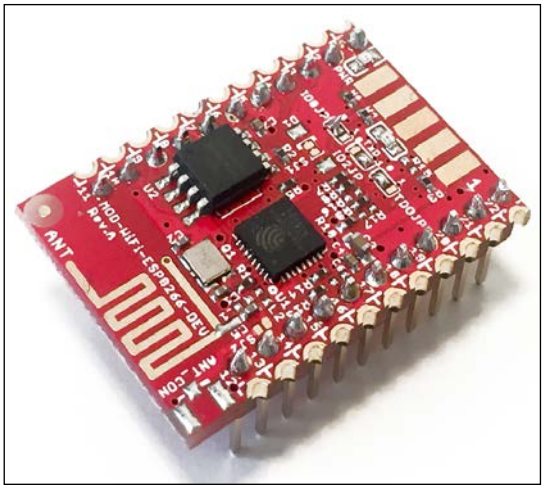
Этот модуль самый известный, так как он действительно маленький и стоит всего 5 долларов. Однако количество доступных выводов GPIO (входных / выходных) весьма ограничено. Его также сложно подключить к стандартной макетной плате.

Если вы выберете этот модуль, в этой книге есть несколько проектов, которые вы, возможно, не сможете выполнить. Например, вы не сможете выполнять проекты с использованием аналоговых датчиков, поскольку аналоговый входной вывод недоступен.

Вы можете найти дополнительную информацию об этом модуле по адресу:

`https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf`

Но на рынке есть много других модулей, которые дают вам доступ ко всем контактам ESP8266. Например, мне очень нравится модуль ESP8266 Olimex, который к тому же дешев (около 10 долларов):

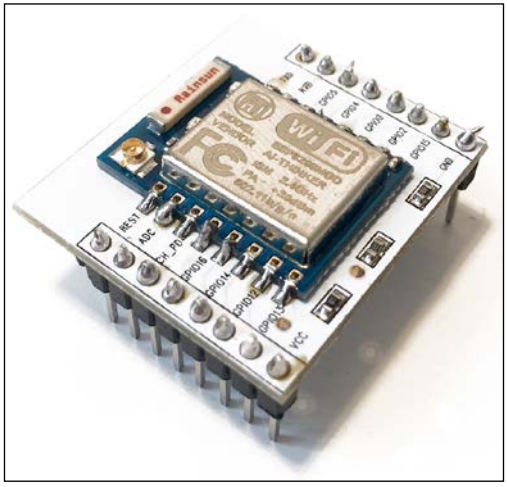


Этот модуль можно легко установить на макетную плату, и вы легко получите доступ ко всем контактам ESP8266. Я буду использовать его в большей части этой книги, и поэтому я рекомендую использовать аналогичный модуль.

Вы можете найти дополнительную информацию об этом модуле по адресу:

`https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware`

Еще один вариант - использовать плату на основе ESP-12, которая является версией ESP8266, предназначенной для интеграции на печатные платы. Эта версия также дает вам доступ ко всем контактам ESP8266. Найти коммутационные платы для этого чипа относительно несложно. Например, это плата, которую я купил на Tindie:



Вы можете найти дополнительную информацию об этом модуле на:

```
http://www.seeedstudio.com/wiki/images/7/7d/ESP-12E_brief_spec.pdf
```

Вы также можете получить коммутационную плату Adafruit ESP8266, которая объединяет ESP-12:

```
http://www.adafruit.com/product/2471
```

Другое решение - использовать комплект разработчика NodeMCU, который похож на плату Olimex, но также имеет встроенный преобразователь USB-to-Serial, а также бортовой источник питания. Ее легче было использовать, но в то время, когда писалась эта книга, ее было трудно найти. Вы можете получить дополнительную информацию на веб-сайте NodeMCU:

```
http://nodemcu.com/index_en.html
```

Обратите внимание, что с модулем NodeMCU вам нужно будет перевести выводы из модуля в выводы, определенные в ESP8266 Arduino IDE, которую мы собираемся использовать. Соответствие между контактами вы найдете здесь:

```
https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en#new_
gpio_map
```

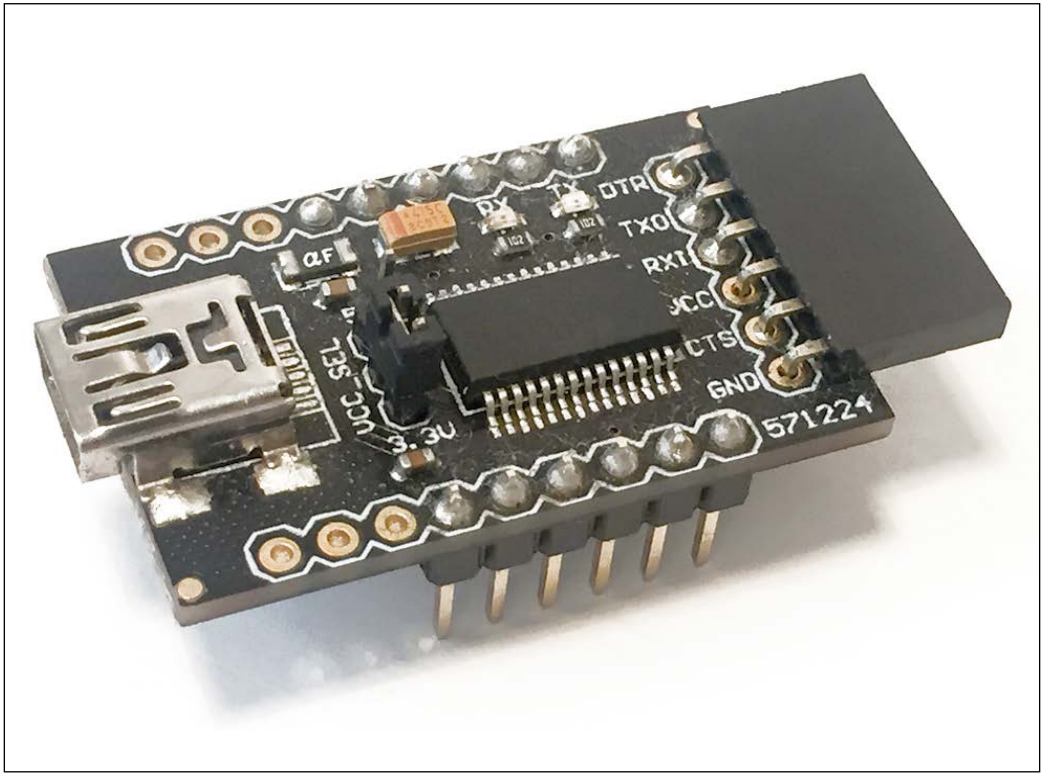
Аппаратные требования

Давайте теперь посмотрим, что нам нужно, чтобы чип ESP8266 работал. Во-первых, вам понадобится способ запрограммировать ESP8266. Вы можете использовать для этого плату Arduino, но для меня действительно замечательным в ESP8266 является то, что она может работать полностью автономно, используя встроенный процессор.

Итак, чтобы запрограммировать чип, я буду использовать программатор USB FTDI.

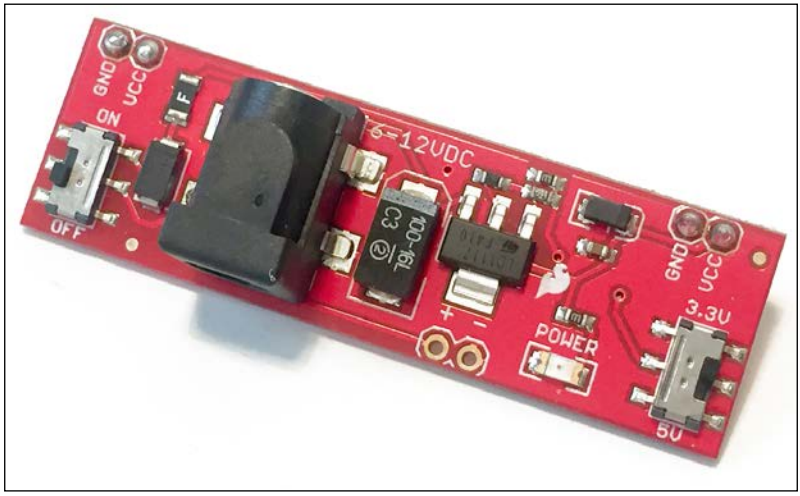
Обратите внимание, что он должен быть совместим с логическим уровнем микросхемы ESP8266, по питанию 3,3 В.

Я использовал модуль, который можно переключать между 3,3 В и 5 В:



Вам также понадобится специальный блок питания для питания микросхемы. Это момент, о котором часто забывают, и он приводит к множеству проблем. Если вы, например, пытаетесь запитать микросхему ESP8266 от 3,3 В, поступающего с платы FTDI или от платы Arduinoboard, она просто не будет работать правильно.

Следовательно, для большинства модулей ESP8266 вам понадобится специальный источник питания, который может обеспечить не менее 300 мА. Некоторые платы имеют встроенный порт micro-USB и стабилизатор напряжения, который может обеспечить необходимый ток для ESP8266, но это не относится к плате, которую мы будем использовать в этой первой главе. Я использовал блок питания на плате, который может выдавать до 500 мА при 3,3 В:



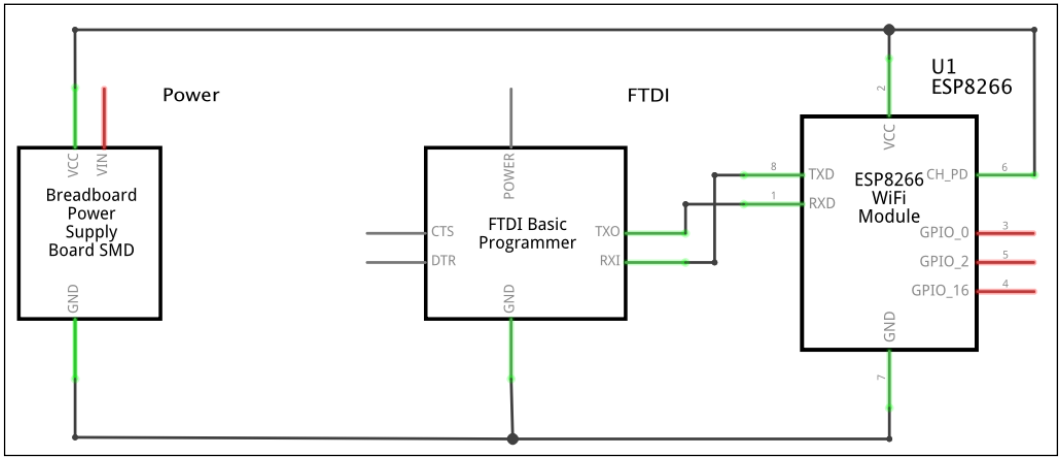
Это список всех компонентов, которые вам понадобятся для использования микросхемы ESP8266:

)

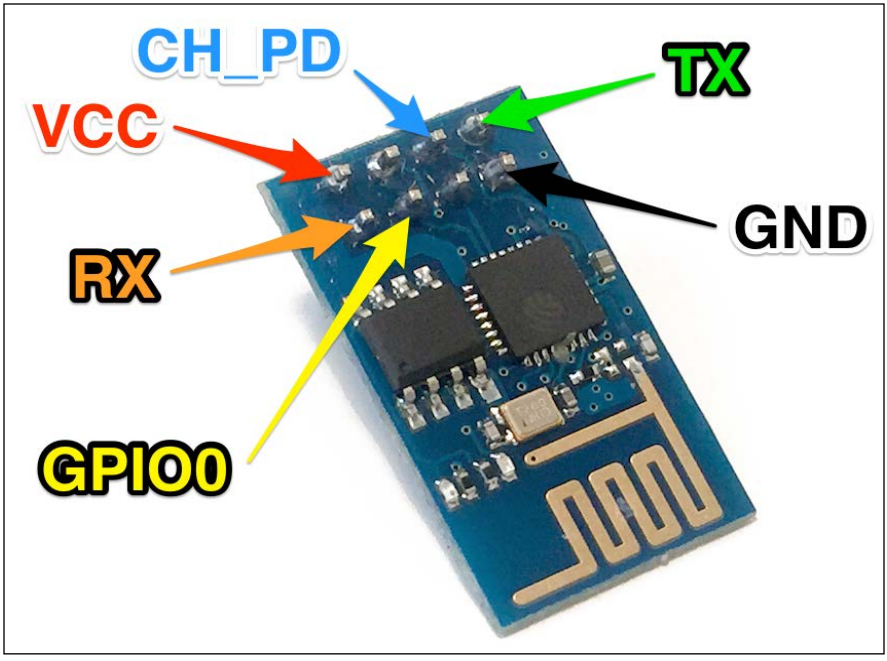
- Модуль ESP8266 Olimex
- Источник питания макетной платы 3,3 В
- Модуль USB 3.3V FTDI
- Макетная плата
- Перемычки

Аппаратная конфигурация

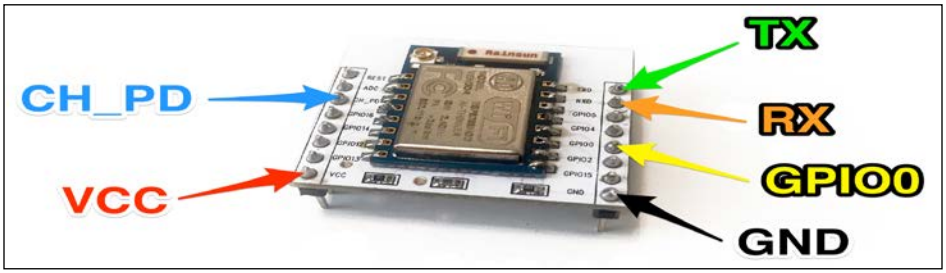
Теперь мы рассмотрим способ настройки устройства для первого использования вашей платы ESP8266. Вот как мы соединяем разные компоненты:



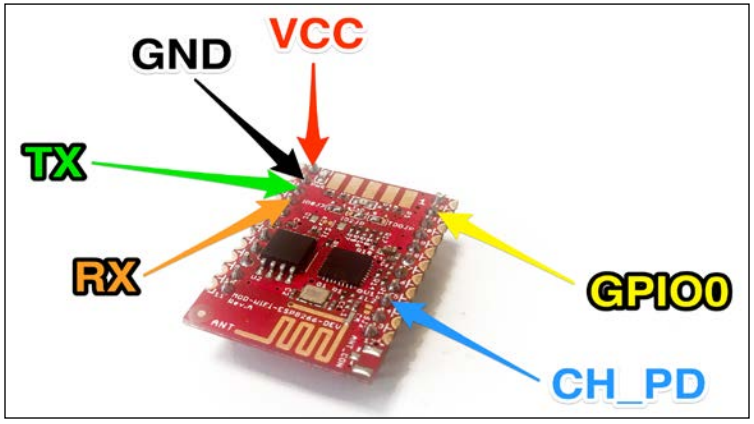
В зависимости от используемой платы контакты могут иметь разные названия. Поэтому я создал картинки, чтобы помочь вам с каждым модулем. Вот те контакты, которые вам понадобятся на маленькой плате ESP:



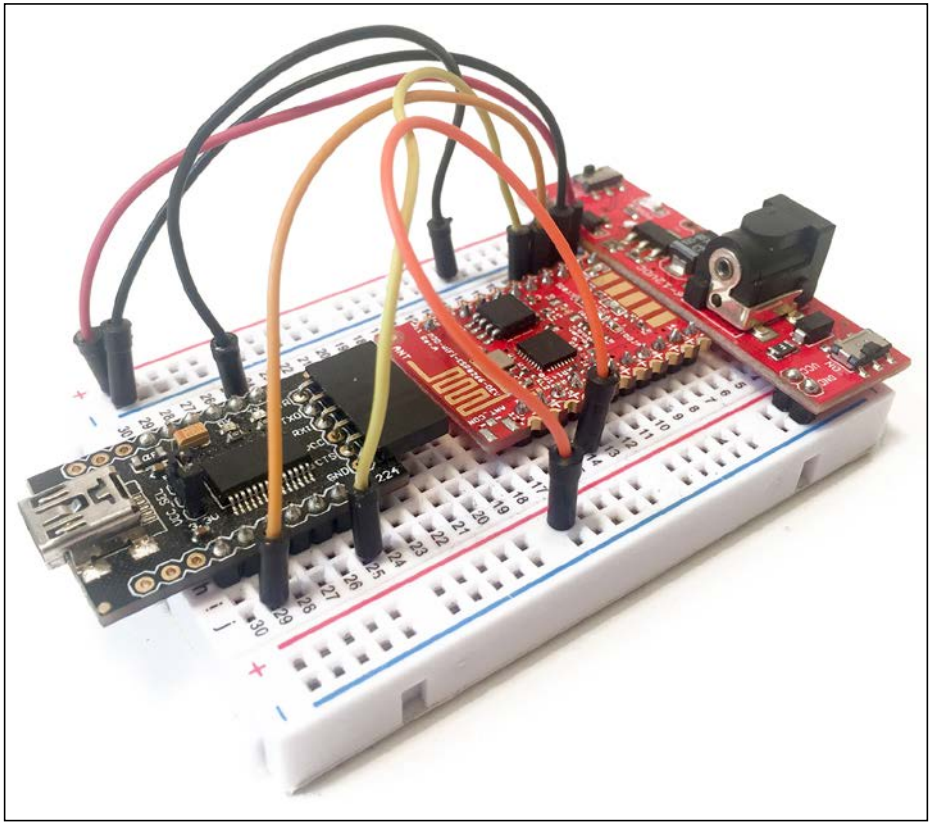
То же самое и с платой ESP-12, установленной на макетном адаптере:



Наконец, это изображение платы Olimex:



Вот так плата Olimex будет выглядеть в нашем проекте:



Убедитесь, что все подключения правильны

Также убедитесь, что все переключатели ваших компонентов (Модуль FTDI и источник питания) установлены на 3,3 В, иначе будет поврежден ваш чип.

Также подключите один провод к контакту GPIO 0 ESP8266. Пока не подключайте его ни к чему другому, но он понадобится вам позже, чтобы перевести чип в режим программирования.

Установка Arduino IDE для ESP8266

Теперь, когда мы полностью настроили оборудование для ESP8266, мы готовы настроить его с помощью Arduino IDE.

Самый простой способ использовать модуль ESP8266 - использовать последовательные команды, поскольку чип в основном представляет собой приемопередатчик Wi-Fi / последовательного интерфейса. Однако это неудобно, и я не рекомендую это делать.

Я рекомендую просто использовать Arduino IDE, которую вам нужно будет установить на свой компьютер. Это делает использование микросхемы ESP8266 очень удобным, поскольку мы будем использовать хорошо известную IDE Arduino, так что это метод, который мы будем использовать во всей книге.

Теперь мы собираемся настроить ваш чип ESP8266 с помощью Arduino IDE. Это отличный способ использовать чип, так как вы сможете запрограммировать его с помощью хорошо известной IDE Arduino, а также повторно использовать несколько существующих библиотек Arduino.

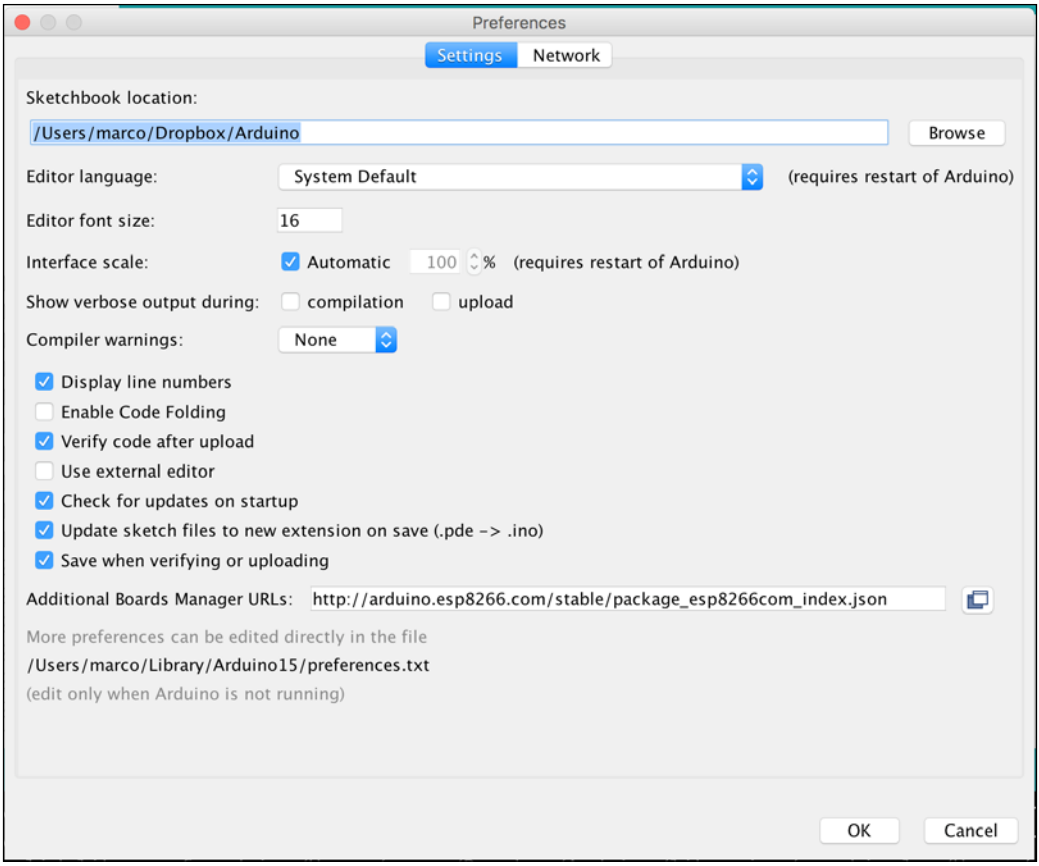
Если это еще не сделано, установите последнюю версию Arduino IDE. Вы можете получить его по адресу http://www.arduino.cc/en/main/software.

Теперь вам нужно выполнить следующие шаги, чтобы иметь возможность настроить ESP8266 с помощью Arduino IDE:

1. Запустите Arduino IDE и откройте окно настроек.

2. Введите следующий URL-адрес в поле URL-адреса дополнительного менеджера платы:
   `http://arduino.esp8266.com/stable/package_esp8266com_index.json`

3. Откройте Boards Manager из меню Инструменты | плата и установите платформу esp8266, как показано здесь:



Подключение вашего модуля к вашей сети Wi-Fi

Теперь мы собираемся проверить, правильно ли работают ESP8266 и Arduino IDE, и подключим ваш чип к локальной сети Wi-Fi.

Для этого выполним следующие шаги:

1. Сначала нам нужно написать код, а затем загрузить его на плату. Код прост; мы просто хотим подключиться к локальной сети Wi-Fi и распечатать IP-адрес платы. Это код для подключения к сети:

```
// Импорт необходимых библиотек
#include <ESP8266WiFi.h>

// WiFi parameters
```

```
constchar* ssid = "your_wifi_name";
constchar* password = "your_wifi_password";

void setup(void)
{
// Start Serial
Serial.begin(115200);
// Connect to WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected");
// Print the IP address
Serial.println(WiFi.localIP());
}

void loop() {

}
```

Введите в код собственное имя Wi-Fi и пароль. Сохраните этот файл под любым именем.

2. Теперь перейдите в Инструменты | Платы и выберите Generic ESP8266 Module. Также выберите соответствующий последовательный порт.

3. После этого нам нужно перевести плату в режим загрузчика, чтобы мы могли ее запрограммировать. Для этого подключите контакт GPIO 0 к земле через кабель, который мы подключили к GPIO 0. Затем выключите и включите питание платы.

4. Теперь загрузите код на плату и откройте монитор последовательного порта, когда это будет сделано и установите его скорость на 115200. Теперь отсоедините кабель между GPIO 0 и GND, выключите и включите плату. Вы должны увидеть следующее сообщение

```
WiFi connected
192.168.1.103
```

Если вы видите это сообщение и IP-адрес, поздравляем, ваша плата теперь подключена к вашей сети Wi-Fi! Теперь вы готовы создавать свои первые проекты с использованием микросхемы ESP8266.

Резюме

В этой первой главе книги мы узнали основы работы с ESP8266.

Сначала мы узнали обо всех различных платах, доступных для ваших проектов ESP8266. Затем мы увидели, как подключить модули ESP8266. Наконец, мы увидели, как установить IDE Arduino и настроить ее для ESP8266, и закончили главу, фактически загрузив очень простой скетч в ESP8266.

В следующей главе мы собираемся использовать инструменты, которые мы настроили, и построить несколько базовых проектов с использованием чипа ESP8266 Wi-Fi.

2

Первые проекты с ESP8266

Теперь, когда ваш чип ESP8266 готов к использованию и вы можете подключить его к своей сети Wi-Fi, мы можем создавать с его помощью некоторые базовые проекты. Это поможет вам понять основы ESP8266.

В этой главе мы увидим три проекта: как управлять светодиодом, как читать данные с вывода GPIO и как получать содержимое с веб-страницы. Мы также увидим, как считывать данные с цифрового датчика.

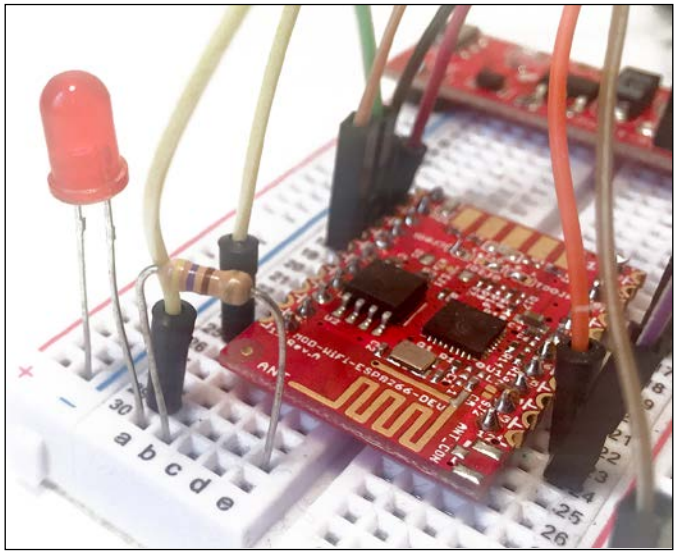
Управление светодиодом

Сначала мы посмотрим, как управлять простым светодиодом. Выводы GPIO ESP8266 могут быть настроены для реализации многих функций: входов, выходов, выходов PWM, а также обмена данными по протоколам SPI или I2C. Этот первый проект научит вас использовать GPIO-выводы микросхемы вкачестве выходов:

1. Первый шаг - добавить в наш проект светодиод. Вот дополнительные компоненты, которые вам понадобятся для этого проекта:
    - 5 мм светодиод
    - Резистор 330 Ом для ограничения тока светодиода
2. Следующим шагом будет подключение светодиода с резистором к плате ESP8266.
3. Затем поместите светодиод на макетную плату, подключив самый длинный вывод светодиода (анод) к одному выводу резистора.

4. Затем подключите другой конец резистора к контакту 5 GPIO ESP8266, а другой конец светодиода - к земле.

   Вот как это должно выглядеть в конце:



5. Теперь мы собираемся зажечь светодиод, запрограммировав чип ESP8266, так же, как мы это делали в первой главе книги, подключив его к сети Wi-Fi.

   Это полный код этого раздела:

```
// Импортировать необходимые библиотеки
#include <ESP8266WiFi.h>

void setup() {

// Установите GPIO 5 как выход
pinMode(5, OUTPUT);

// GPIO 5 установлен в ВЫСОКОЕ состояние
digitalWrite(5, HIGH);

}
void loop() {

}
```

Этот код просто устанавливает вывод GPIO в качестве выхода, а затем применяет к нему состояние HIGH. Состояние HIGH означает, что вывод активен и на него подается положительное напряжение (3,3 В). НИЗКОЕ состояние будет означать, что на выходе 0 В.

6. Теперь вы можете скопировать этот код и вставить его в IDE Arduino.
7. Затем загрузите код на плату, следуя инструкциям из предыдущей главы. Вы должны сразу увидеть, что светодиод зажегся. Вы можете снова выключить его, используя в коде digitalWrite (5, LOW). Вы также можете, например, изменить код, чтобы ESP8266 включал и выключал светодиод каждую секунду.

Чтение данных с вывода GPIO

Во втором проекте этой главы мы собираемся прочитать состояние вывода GPIO. Для этого воспользуемся той же выводом, что и в предыдущем проекте. Поэтому вы можете удалить светодиод и резистор, которые мы использовали в предыдущем проекте.

Теперь просто подключите этот контакт (GPIO 5) платы к положительному источнику питания на макетной плате с помощью провода, подав на этот контакт сигнал 3,3 В.

Считывание данных с вывода действительно просто. Это полный код для этой части:

```
// Импортировать необходимые библиотеки
#include <ESP8266WiFi.h>

void setup(void)
{
// Start Serial (показ результатов на последовательном мониторе)
Serial.begin(115200);

// Установите GPIO 5 в качестве входа
pinMode(5, INPUT);}void loop() {
// Прочтите GPIO 5 и распечатайте его через последовательный порт
Serial.print("State of GPIO 5: ");
Serial.println(digitalRead(5));

// ждем 1 секунду
     delay(1000);
}
```

Мы просто устанавливаем вывод как вход, считываем значение этого вывода и распечатываем его каждую секунду. Скопируйте и вставьте этот код в IDE Arduino, затем загрузите его на плату, следуя инструкциям из предыдущей главы.

Вот результат, который вы должны получить на последовательном мониторе:

```
State of GPIO 5: 1
```

Мы видим, что возвращаемое значение - 1 (цифровое состояние HIGH), что мы и ожидали, потому что мы подключили контакт к положительному источнику питания. В качестве теста вы также можете подключить контакт к земле, и состояние должно перейти в 0.

Получение контента с веб-страницы

В качестве последнего проекта в этой главе мы, наконец, собираемся использовать Wi-Fi соединение чипа для захвата содержимого страницы. Мы просто воспользуемся страницей www.example.com, поскольку это базовая страница, которая в основном используется в тестовых целях.

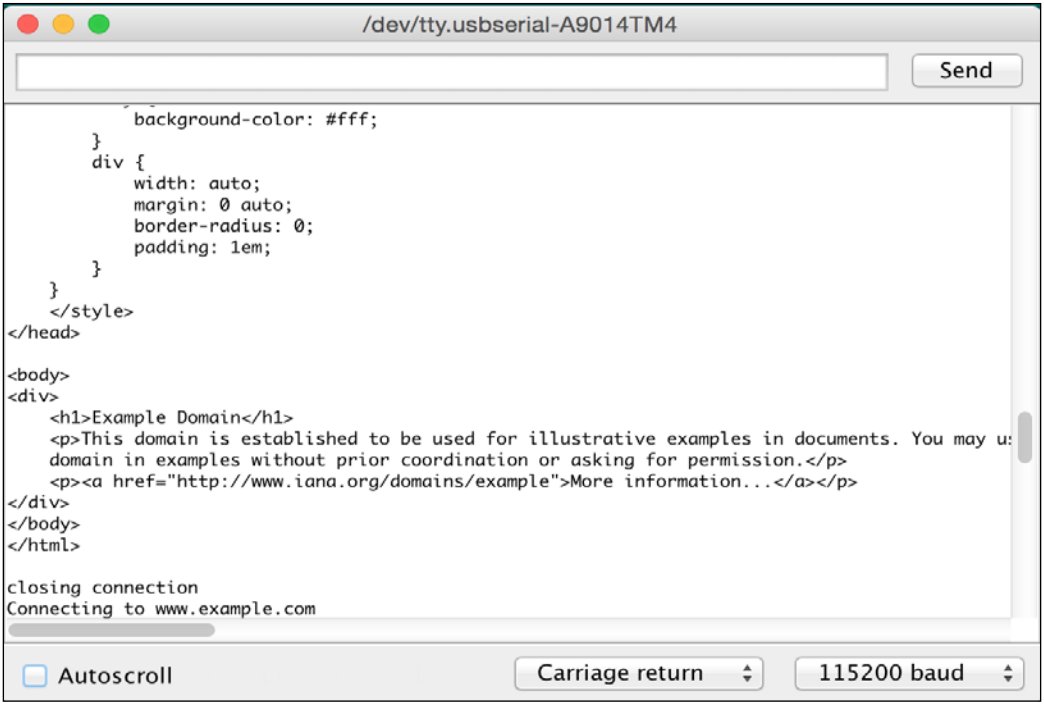
Это полный код этого проекта:

```
// Импортировать необходимые библиотеки
#include <ESP8266WiFi.h>

// Параметры WiFi
constchar* ssid = "your_wifi_network";
constchar* password = "your_wifi_password";

// Host
constchar* host = "www.example.com";

void setup() {
// Start Serial
Serial.begin(115200);
// Начнем с подключения к сети Wi-Fi
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
Serial.print(".");
  }
```

```
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

int value = 0;

void loop() {

Serial.print("Connecting to ");
Serial.println(host);

// Используйте класс клиента WiFi для создания TCP-соединений
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
  }

// Это отправит запрос на сервер
client.print(String("GET /") + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
  delay(10);

// Прочитать ответ от сервера и распечатать в послед. мониторе
while(client.available()){
    String line = client.readStringUntil('\r');
Serial.print(line);
  }

Serial.println();
Serial.println("closing connection");
  delay(5000);

}
```

Код действительно прост: сначала мы открываем соединение с сайтом example.com, а затем отправляем запрос GET, чтобы получить содержимое страницы. Используя while (client.available ()), мы также прослушиваем входящие данные и распечатываем их в последовательном мониторе.

Теперь вы можете скопировать этот код и вставить его в IDE Arduino. Затем загрузите его на плату, следуя инструкциям из главы 1 «Начало работы с ESP8266» в разделе «Подключение модуля к сети Wi-Fi». Вот что вы должны увидеть на последовательном мониторе:



Это в основном содержимое страницы в чистом HTML-коде.

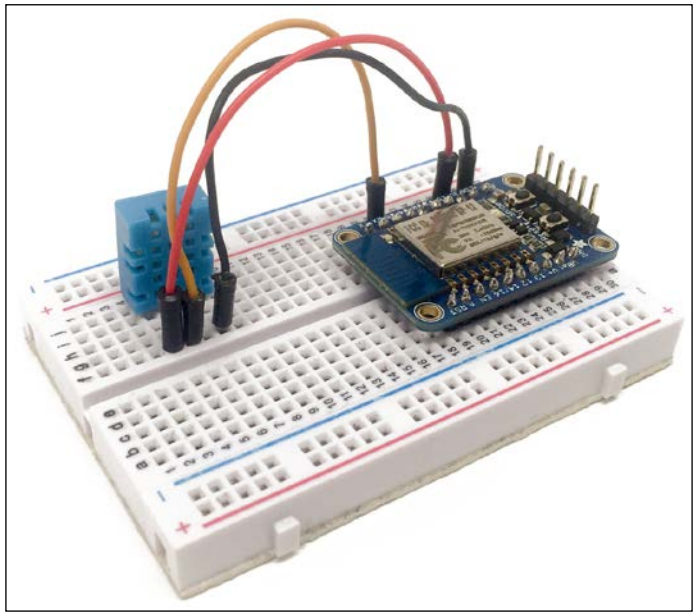
Считывание данных с цифрового датчика

В этом последнем разделе этой главы мы собираемся подключить цифровой датчик к нашему чипу ESP8266 и считать с него данные. В качестве примера мы будем использовать датчик DHT11, который можно использовать для измерения температуры и влажности окружающей среды.

Давайте теперь подключим этот датчик к вашему ESP8266:

1. Сначала поместите датчик на макетную плату. Затем подключите первый контакт датчика к VCC, второй контакт к контакту 5 ESP8266, а четвертый контакт датчика к GND.

   Вот как это будет выглядеть в конце:



Обратите внимание, что здесь я использовал другую плату ESP8266, коммутационную плату Adafruit ESP8266. Я буду использовать эту плату в нескольких главах этой книги.

   В этом примере мы также будем использовать структуру aREST, чтобы вы могли легко получить удаленный доступ к измерениям. aREST - это полноценный фреймворк для удаленного управления вашими платами ESP8266 (в том числе из облака), и мы собираемся использовать его несколько раз в книге. Дополнительную информацию об этом можно найти по следующему URL-адресу: http://arest.io/.

2. Теперь сконфигурируем плату. Код слишком длинный, чтобы его можно было здесь вставить, но сейчас я подробно расскажу о его наиболее важной части.

3. Он начинается с включения необходимых библиотек:

```
#include "ESP8266WiFi.h"
#include <aREST.h>
#include "DHT.h"
```

4. Чтобы установить эти библиотеки, просто найдите их в диспетчере библиотек Arduino IDE. Далее нам нужно установить контакт, к которому подключен датчик DHT:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

5. После этого мы объявляем экземпляр датчика DHT:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

6. Как и раньше, вам нужно будет вставить в код собственное имя Wi-Fi и пароль:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

7. Определяем две переменные, которые будут содержать измерения датчика:

```
float temperature;
float humidity;
```

8. В функции скетча setup () мы инициализируем датчик:

```
dht.begin();
```

9. В функции setup () мы предоставляем переменные REST API, чтобы мы могли получить к ним удаленный доступ через Wi-Fi:

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

10. Наконец, в функции loop () мы производим измерения с датчика:

```
humidity = dht.readHumidity();
temperature = dht.readTemperature();
```

11. Пришло время протестировать проект! Просто возьмите весь код из папки с кодами и поместите его в IDE Arduino. Также не забудьте установить библиотеку laTEST Arduino с помощью диспетчера библиотек Arduino.

12. Теперь переведите плату ESP8266 в режим загрузчика и загрузите код на плату. После этого перезагрузите плату и откройте монитор последовательного порта. Вы должны увидеть IP-адрес отображаемой платы:

13. Теперь мы можем получить доступ к измерениям с датчика удаленно. Просто зайдите в свой любимый веб-браузер и введите:

```
192.168.115.105/temperature
```

Вы должны сразу получить ответ с платы с отображением температуры:

```
{
  "temperature": 25.00,
"id": "1",
"name": "esp8266",
"connected": true
}
```

Конечно, вы можете сделать то же самое с влажностью.

Обратите внимание, что здесь мы использовали aREST API, который мы будем использовать в нескольких других главах этой книги. Вы можете узнать об этом на http://arest.io/.

Поздравляем, вы только что завершили свои самые первые проекты с использованием чипа ESP8266! Не стесняйтесь экспериментировать с тем, что вы узнали в этой главе, и начните больше узнавать о том, как настроить свой чип ESP8266.

Резюме

В этой главе мы реализовали наши первые базовые проекты с использованием чипа ESP8266 Wi-Fi.
Сначала мы узнали, как управлять простым выходом, контролируя состояние светодиода. Затем мы увидели, как считывать состояние цифрового вывода на микросхеме. Наконец, мы узнали, как считывать данные с цифрового датчика и фактически получать эти данные с помощью структуры aREST, которую мы будем использовать в нескольких главах этой книги.

В следующей главе мы перейдем непосредственно к основной теме книги и создадим наш первый проект Интернета вещей с использованием ESP8266.

3

Регистрация облачных данных с помощью ESP8266

В этой главе мы собираемся использовать ESP8266 для автоматического протоколирования измерений температуры и влажности в облаке и отображения этих измерений на онлайн-панели инструментов.

Следуя этому проекту, вы сможете создать небольшую и дешевую измерительную платформу, которая регистрирует данные в облаке. Конечно, это можно применить к нескольким типам датчиков, например к датчикам движения.

Аппаратные и программные требования

Для этого проекта вам понадобится следующие компоненты:

- Конечно, вам понадобится микросхема ESP8266. Вы можете, например, использовать модуль Olimex ESP8266.
- Также вам понадобится датчик температуры. Я использовал датчик DHT11, который очень прост в использовании и позволяет нам измерять температуру и влажность окружающей среды.
- Вам также понадобится USB-модуль FTDI 3,3 В для программирования микросхемы ESP8266.
  Наконец, вам также понадобятся перемычки и макетная плата.

Это список всех компонентов, которые будут использоваться в этой главе, а также источники, где вы можете их приобрести:

- Модуль ESP8266 Olimex
- Источник питания макетной платы 3,3 В
- Модуль USB 3.3V FTDI
- Датчик DHT11
- Макетная плата
- Перемычки

Что касается программы, вам понадобятся:

- Последняя версия IDE Arduino, которую вы можете получить: `http://www.arduino.cc/en/Main/Software`.

Выполните следующую процедуру, чтобы добавить плату ESP8266 в IDE Arduino:

1. Запустите IDE Arduino и откройте окно настроек.
2. Введите URL-адрес в поле URL-адреса Additional Board Manager:

   ```
   http://arduino.esp8266.com/package_esp8266com_index.json
   ```

3. Откройте Boards Manager, перейдя в Инструменты | Платы menu и установите платформу esp8266.
4. Также понадобится библиотека DHT, которую вы можете получить по адресу:

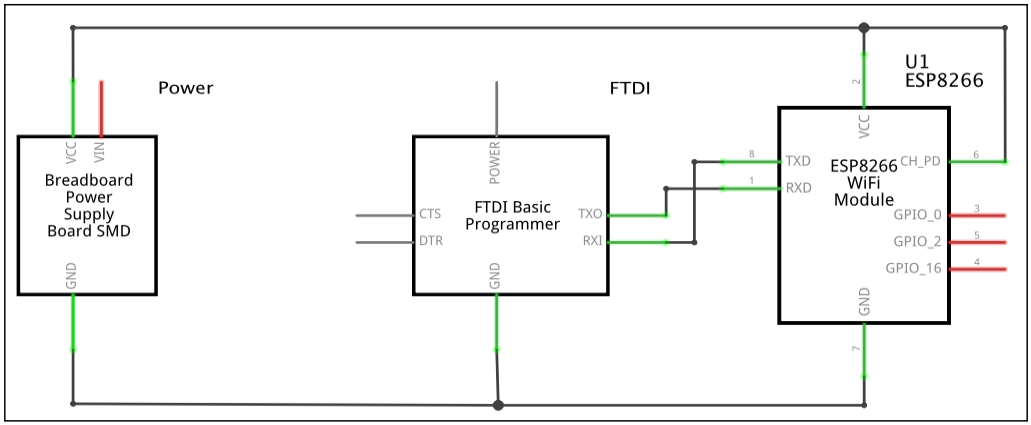
   ```
   https://github.com/adafruit/DHT-sensor-library
   ```

Чтобы установить библиотеку Arduino:

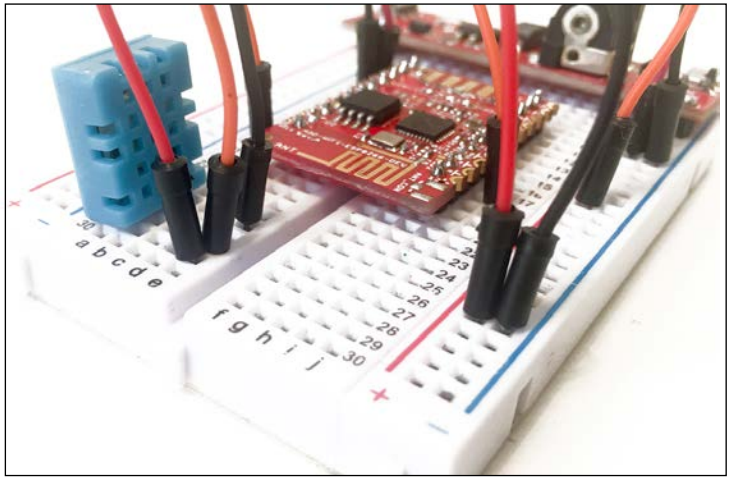
1. Сначала загрузите библиотеку из репозитория GitHub.
2. Затем войдите в IDE Arduino и перейдите в Sketch | Включить библиотеку | Добавить библиотеку .ZIP.
3. Наконец, выберите файл, который вы только что загрузили.

Аппаратная конфигурация

Сначала мы посмотрим, как произвести аппаратную конфигурацию для использования платы ESP8266.



1. По сути, вам необходимо подключить источник питания макетной платы VCC и GND к ESP8266 VCC и GND. Также подключите контакт GND платы преобразователя FTDI к GND ESP8266.
2. Затем подключите TX от платы FTDI к RX платы ESP8266, а затем RX к TX.
3. Наконец, подключите вывод CH_PD (или CHIP_EN) платы ESP8266 к VCC.
4. Затем, просто поместите датчик DHT11 на макетную плату.
5. Затем подключите левый контакт к VCC (красная шина питания), правый контакт к GND (синяя шина питания), а контакт рядом с VCC к контакту 5 GPIO на микросхеме ESP8266.
   Это окончательный результат, без кабелей FTDI USB-Serial:



Убедитесь, что вы подключили все по схемам. Также убедитесь, что модуль FTDI и модуль источника питания установлены на 3,3 В, иначе можно повредить ваш чип.

6. Также подключите один провод к контакту 0 GPIO ESP8266. Пока не подключайте его ни к чему другому, но он понадобится вам позже, чтобы перевести чип в режим программирования.

Тестирование датчика

Теперь воспользуемся датчиком. Опять же, помните, что мы используем IDE Arduino, поэтому мы можем кодировать так же, как если бы это было на плате Arduino. Здесь мы просто напечатаем значение температуры внутри последовательного монитора Arduino IDE. Если это еще не было сделано, установите библиотеку датчиков Adafruit DHT с помощью диспетчера библиотек Arduino IDE.

Это полный код для этой части:

```
// Libraries
#include "DHT.h"

// Pin
#define DHTPIN 5

// Use DHT11 sensor
#define DHTTYPE DHT11

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE, 15);

void setup() {

// Start Serial
Serial.begin(115200);

// Init DHT
dht.begin();
}

void loop() {
```

```
// Считывание температуры и влажности
float h = dht.readHumidity();
// Read temperature as Celsius
float t = dht.readTemperature();

// Отображение данных
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.println(" *C ");

// ждите 2 секунды между измерениями.
delay(2000);

}
```

Посмотрим детали кода. Вы можете видеть, что вся измерительная часть содержится внутри функции loop (), которая заставляет код внутри нее повторяться каждые 2 секунды.

Затем мы считываем данные с датчика DHT11 и печатаем значение температуры и влажности в мониторе последовательного порта.

Полный код находится в папке с кодами

Теперь давайте начнем с шагов по тестированию датчика:

1. Вставьте предыдущий код в IDE Arduino. Затем перейдите в Инструменты | Платы и выберите Generic ESP8266 Module. Также выберите соответствующий последовательный порт, который соответствует используемому конвертеру FTDI.
2. После этого нам нужно перевести плату в режим загрузчика, чтобы мы могли ее запрограммировать. Для этого подключите контакт GPIO 0 к земле через провод, который мы ранее подключили к GPIO 0.
3. Затем выключите и снова включите питание платы.

4. Теперь загрузите код на плату и откройте монитор последовательного порта, когда это будет сделано. Также установите скорость последовательного монитора на 115200.
5. Теперь отсоедините кабель между GPIO 0 и GND и снова выключите и снова включите плату.

Вы должны немедленно увидеть показания температуры и влажности внутри последовательного монитора. Когда я тестировал мой датчик, он показывал около 24 градусов по Цельсию, что является реалистичным значением.

Запись данных в Dweet.io

Теперь мы посмотрим, как регистрировать измерения температуры и влажности в облаке. Воспользуемся облачным сервисом Dweet.io здесь, что очень удобно для логирования.

```
http://dweet.io/
```

Поскольку код для этой части очень длинный, мы рассмотрим только важные части . Конечно, вы можете получить весь код из папки с кодами

Опять же, все измерения выполняются внутри функции loop () скетча, которая заставляет код повторяться каждые 10 секунд, используя функцию delay ().
Внутри этого цикла мы подключаемся к серверу Dweet.io с помощью:

```
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
}
```

Затем мы считываем данные с датчика с помощью:

```
int h = dht.readHumidity();
int t = dht.readTemperature();
```

После этого мы отправляем данные на сервер Dweet.io с помощью:

```
client.print(String("GET /dweet/for/myesp8266?temperature=") +
  String(t) + "&humidity=" + String(h) + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
```

Возможно, вы захотите заменить здесь myesp8266 - имя вашего устройства на Dweet.io. Используйте сложное имя (например, пароль), чтобы убедиться, что вы создаете уникальное устройство на Dweet.io.

Мы также печатаем любые данные, полученные через последовательный порт, с помощью:

```
while(client.available()){
  String line = client.readStringUntil('\r');
Serial.print(line);
}
```

Обратите внимание, что вам также необходимо изменить код, чтобы вставить собственное имя сети Wi-Fi и пароль. Теперь вы можете загрузить код на плату ESP8266, используя инструкции, которые мы видели ранее, и открыть монитор последовательного порта.

Вы должны увидеть, что каждые 10 секунд запрос отправляется на сервер Dweet.io, и вы получаете ответ:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 147
Date: Mon, 16 Mar 2015 10:03:37 GMT
Connection: keep-alive

{"this":"succeeded","by":"dweeting","the":"dweet",
"with":{"thing":"myesp8266","created":"2015-03-16T10:03:37.053Z",
"content":{"temperature":24, "humidity":39}
  }
}
```

Если вы видите сообщение об успешном завершении, поздравляем, вы только что зарегистрировали данные в облаке с помощью своего чипа ESP8266!

Отображение данных с помощью Freeboard.io

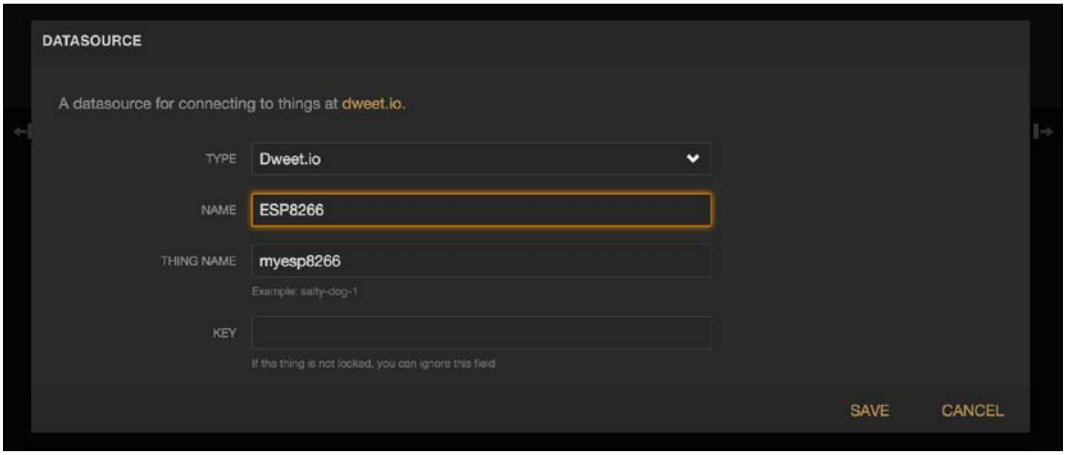
Теперь мы хотели бы отображать записанные данные на панели инструментов, к которой можно получить доступ из любой точки мира. Для этого мы собираемся использовать сервис, который я люблю использовать вместе с Dweet.io: Freeboard.io.

Начнем с использования Freeboard.io:

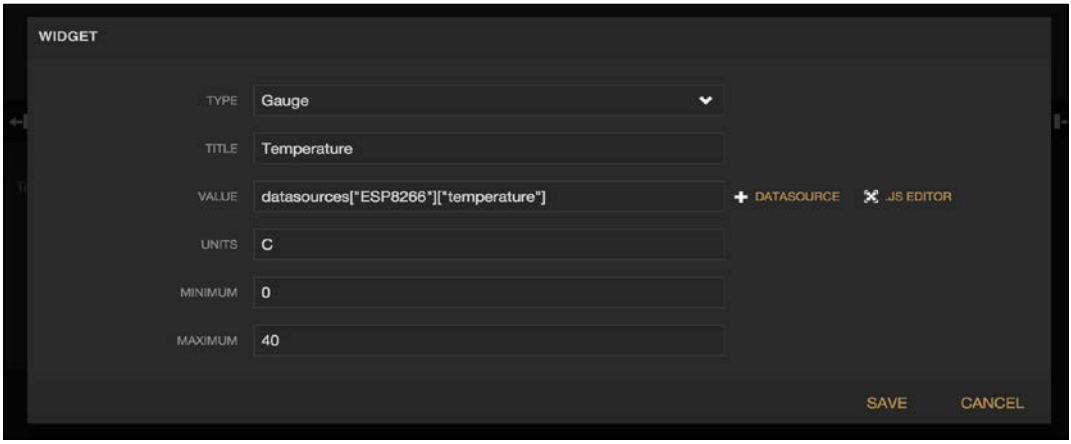
1. Сначала создайте там учетную запись, перейдя по ссылке:

   ```
   https://www.freeboard.io/
   ```

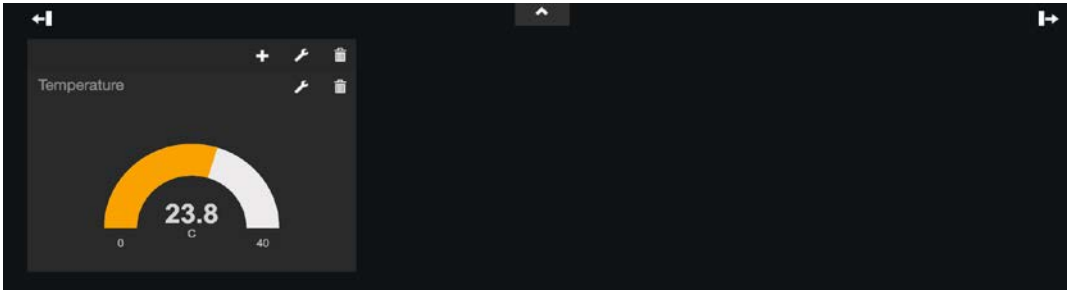
2. Затем создайте новую панель мониторинга и внутри нее новый источник данных. Свяжите этот источник данных со своим Dweet.io, который вы определили в коде ESP8266:



3. Теперь создайте новый виджет Gauge, который мы будем использовать для отображения температуры. Дайте ему имя и свяжите его с температурным полем нашего источника данных:

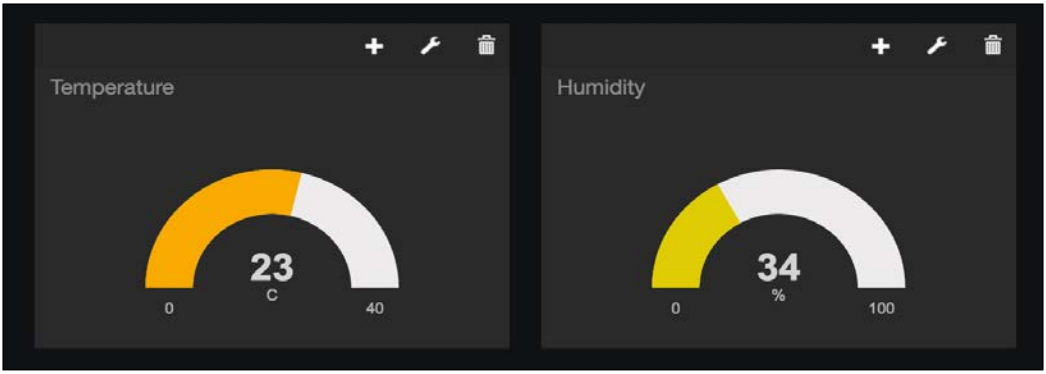


Это конечный результат:

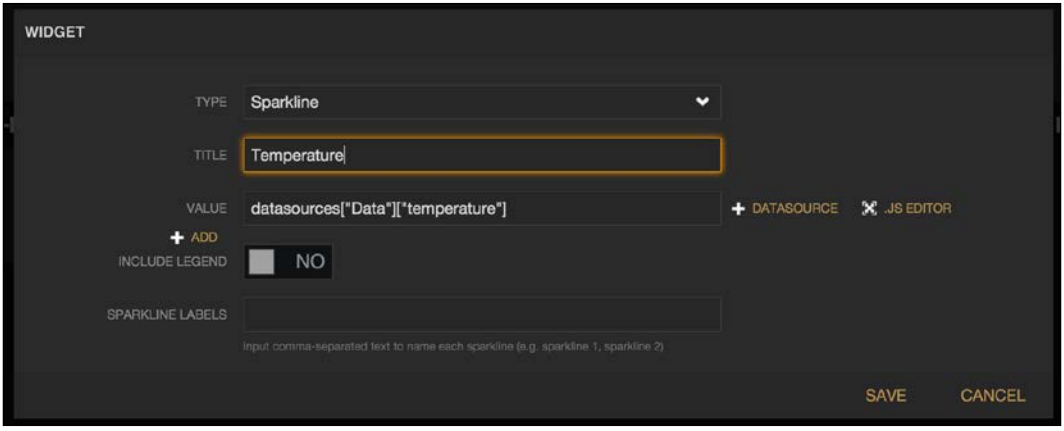


Вы должны увидеть, что данные о температуре, поступающие от ESP8266, регистрируются каждые 10 секунд и сразу отображаются на панели Freeboard.io.
Поздравляем, вы создали очень маленький и дешевый датчик температуры, который регистрирует данные в облаке!

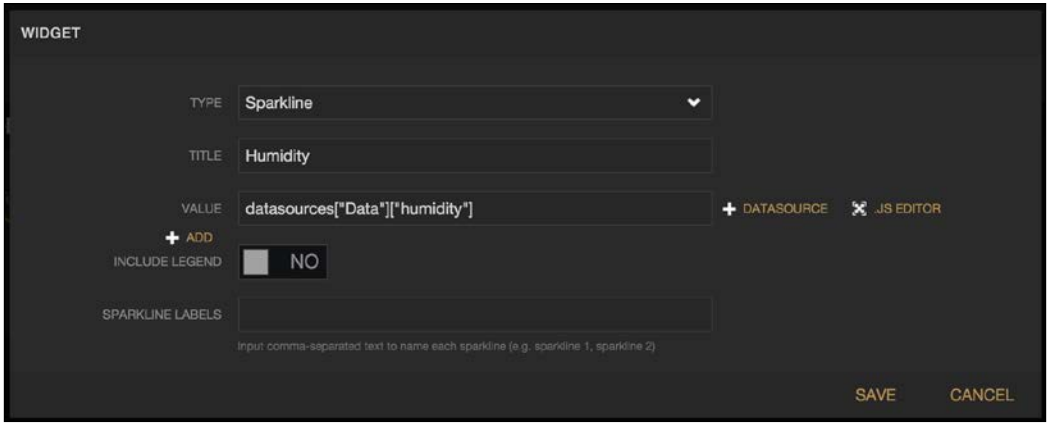
Затем вы можете сделать то же самое с измерениями влажности, а также отобразить их на этой панели:



Также очень легко добавить другие виджеты на панель управления. Например, вы можете захотеть построить историю измерений температуры и влажности. Для этого есть виджет под названием Sparkline. Вы можете создать его так же, как вы создали виджет «Датчик». Начнем с виджета температуры:



Вы не можете сделать то же самое с влажностью:



Это окончательный результат на панели инструментов:

Резюме

Подведем итог тому, что мы сделали в этом проекте. Мы построили простой и дешевый датчик температуры Wi-Fi на базе микросхемы ESP8266. Мы настроили его для автоматической регистрации данных в сервисе Dweet.io и отображали эти измерения на онлайн-панели инструментов.

Есть много способов улучшить этот проект. Вы можете просто добавить в проект дополнительные датчики и отобразить эти измерения на панели инструментов Freeboard.io.

Также можно, например, полностью изменить проект, подключив датчик движения к плате ESP8266. Затем вы можете настроить этот датчик так, чтобы он автоматически отправлял вам оповещение при обнаружении движения, например, по электронной почте или через Twitter.

Возможности безграничны!

В следующей главе мы увидим, как сделать еще одну важную вещь в любом проекте IoT: управлять устройствами из облака.

4

Управляйте устройствами откуда угодно

В предыдущей главе мы увидели, как проводить измерения с помощью микросхемы ESP8266 и отправлять эти данные в Интернет. Мы увидели, как регистрировать эти данные в облаке и отображать их на панели инструментов, к которой можно получить доступ из любого места.

В этой главе мы увидим обратную ситуацию: как мы можем управлять устройствами из любой точки мира? Мы увидим, что это требует использования протокола, отличного от классического HTTP, который называется MQTT.

MQTT может быть трудным в использовании; вот почему мы будем использовать созданную мной библиотеку под названием aREST, которая упростит весь процесс. aREST использует только некоторые из мощных функций MQTT, но этого будет более чем достаточно для управления устройствами из любого места.

В этой главе мы увидим два примера: как уменьшить яркость светодиода и как управлять лампой из любой точки мира. Мы увидим, как это сделать, используя панель управления в облаке.

В конце этого проекта вы сможете управлять лампой, подключенной к вашему чипу ESP8266, простым щелчком мыши на приборной панели, к которой можно получить доступ из любой точки мира.

Аппаратные и программные требования

Для этого проекта вам, конечно же, понадобится микросхема ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266.

Вам также понадобится способ управлять лампой или другими устройствами. Первоначально я использовал простое реле для своих тестов, но быстро перешел на комплект PowerSwitch Tail Kit, который позволяет вам просто и безопасно подключать высоковольтные устройства к вашим проектам.
В качестве светодиода я использовал простой красный светодиод диаметром 5 мм вместе с резистором на 330 Ом.
Вам также понадобится USB-модуль FTDI 3,3 В / 5 В для программирования микросхемы ESP8266.
Наконец, вам также понадобятся перемычки и макетная плата.

Это список всех компонентов, которые будут использоваться в этом руководстве:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- LED
- Резистор 330 Ом
- PowerSwitch Tail Kit
- Макетная плата
- Перемычки

> Обратите внимание, что вам также понадобится устройство для управления. В качестве тестового устройства я использовал простую настольную лампу мощностью 30 Вт, но вы также можете использовать любое другое устройство в своем доме (если номинальная мощность ниже максимальной мощности, принимаемой комплектом PowerSwitch Tail Kit). Вы также можете использовать простое реле или светодиод для тестирования.

Что касается программы, если это еще не сделано, вам нужно будет установить последнюю версию Arduino IDE, которую вы можете получить по адресу:

```
http://www.arduino.cc/en/Main/Software
```

Затем вам также потребуется установить библиотеки aREST и PubSubClient. Чтобы установить библиотеку, просто используйте менеджер библиотек Arduino.

Вам также необходимо создать аккаунт на веб-сайте панели инструментов aREST:

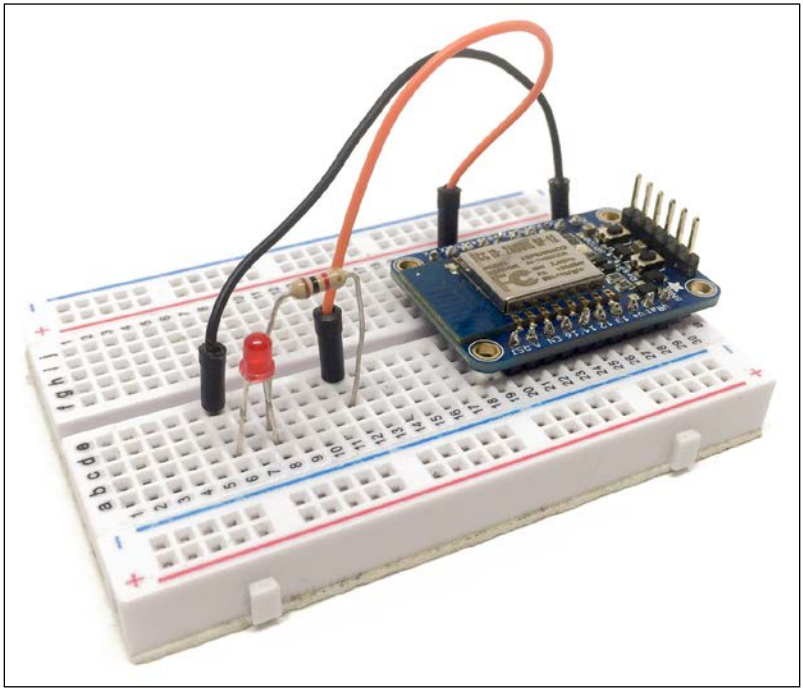
```
http://dashboard.arest.io/
```

Настройка модуля ESP8266 и управление светодиодом

Теперь мы собираемся настроить модуль ESP8266, что означает создание проекта, а также настройку платы, чтобы она могла получать команды из облака.

Просто поместите плату ESP8266 на макетную плату, а затем подключите к ней коммутационную плату FTDI.
Светодиод подключите последовательно с резистором, так чтобы самый длинный вывод светодиода был подключен к резистору. Затем подключите другой вывод резистора к контакту 5 платы ESP8266, а короткий вывод светодиода - к контакту GND.

Это конечный результат:



Теперь мы собираемся настроить плату так, чтобы она могла получать команды из облака.

Это скетч этой части:

```
// Импортировать необходимые библиотеки
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>

// Клиенты
WiFiClient espClient;
PubSubClient client(espClient);

// Создать экземпляр aREST
aREST rest = aREST(client);

// Уникальный ID для идентификации устройства для cloud.arest.io
char* device_id = "9u2co4";

// Параметры WiFi
const char* ssid = "wifi-name";
const char* password = "wifi-password";

// Функции
void callback(char* topic, byte* payload, unsigned int length);

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Установить обратный звонок
  client.setCallback(callback);

  // Дайте имя и ID устройству
  rest.set_id(device_id);
  rest.set_name("devices_control");

  // Подключиться к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
```

```
  Serial.println("WiFi connected");

  // Установить тему вывода
  char* out_topic = rest.get_topic();

}

void loop() {

  // Подключение к облаку
  rest.handle(client);

}

// Обрабатывает сообщение, полученное по темам, на которые вы
подписаны
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

Давайте теперь посмотрим на этот скетч поподробнее.

Он начинается с включения необходимых библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

Затем мы объявляем клиентов Wi-Fi и PubSub:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

После этого мы создаем экземпляр aREST, который позволит плате обрабатывать команды, поступающие из облака:

```
aREST rest = aREST(client);
```

Теперь давайте посмотрим на два важных момента внутри скетча, которые вам нужно будет изменить для вашего собственного проекта.

Первый - это идентификатор устройства ID:

```
char* device_id = "9u2co4";
```

Здесь вам действительно нужно установить собственный идентификатор, так как он будет идентифицировать устройство в сети. На облачном сервере могут быть другие люди, использующие тот же идентификатор, поэтому убедитесь, что вы используете что-то уникальное.

Затем установите имя сети Wi-Fi и пароль:

```
const char* ssid = "wifi-name";
const char* password = "wifi-password";
```

После этого внутри функции скетча setup () мы устанавливаем обратный вызов. Через минуту мы увидим, что это означает, а пока мы просто передаем это экземпляру PubSubClient:

```
client.setCallback(callback);
```

Еще в функции setup () мы устанавливаем идентификатор и имя устройства:

```
rest.set_id(device_id);
rest.set_name("devices_control");
```

В функции скетча loop () мы убеждаемся, что плата всегда пытается подключиться к облачному серверу с помощью:

```
rest.handle(client);
```

Наконец, мы определяем функцию обратного вызова,которая обработает входящие запросы из облака и ответит соответствующим образом:

```
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

Пришло время протестировать наш проект! А пока мы просто убедимся, что он действительно подключен к облачному серверу.

Сначала загрузите код на плату. Убедитесь, что вы изменили идентификатор устройства и настройки Wi-Fi внутри кода. Чтобы загрузить код на плату, следуйте инструкциям, которые мы видели в предыдущих главах этой книги.

Все запросы к доске теперь будут выполняться через облачный сервер aREST, расположенный по адресу:

```
http://cloud.arest.io
```

Чтобы проверить это, перейдите в свой любимый веб-браузер и введите:

```
http://cloud.arest.io/9u2co4/id
```

Конечно, вам нужно заменить ID платы на тот, который вы установили внутри скетча. Он проверит, подключена ли плата к облачному серверу. В этом случае вы должны получить аналогичный ответ в своем веб-браузере:

```
  {
"id": "9u2co4",
"name": "devices_control",
"connected": true
  }
```

Это означает, что наше устройство теперь подключено к сети и отвечает на команды.

Но не будем останавливаться на достигнутом; мы на самом деле собираемся зажечь светодиод из облака.

Во-первых, нам нужно установить вывод 5 как выход. Для этого просто введите:

```
https://cloud.arest.io/9u2co4/mode/5/o
```

В качестве подтверждения вы должны получить следующее сообщение:

```
{
  "message": "Pin D5 set to output",
  "id": "9u2co4",
  "name": "devices_control",
  "connected": true
}
```

Затем включите светодиод с помощью:

```
https://cloud.arest.io/9u2co4/digital/5/1
```

Вы должны сразу увидеть, что светодиод загорелся, и вы также должны получить подтверждающее сообщение в своем браузере:

```
{
  "message": "Pin D5 set to 1",
  "id": "9u2co4",
  "name": "devices_control",
  "connected": true
}
```

Поздравляем, теперь вы можете управлять светодиодом из любой точки мира! Чтобы узнать больше об aREST и командах, которые вы можете использовать, вы можете перейти по ссылке:

```
http://arest.io/
```

Управление LED с облачной панели инструментов

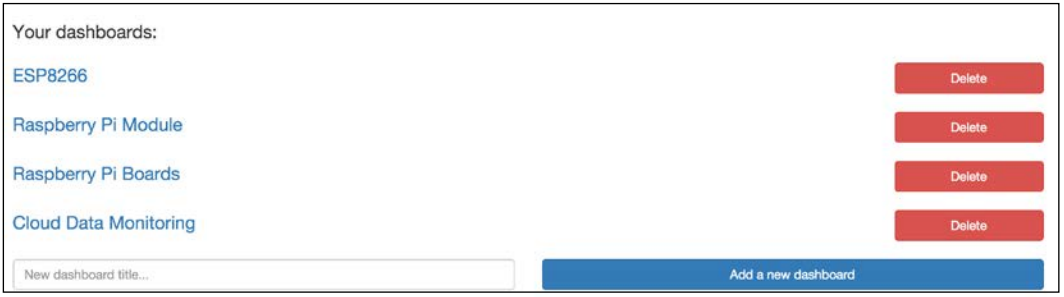
Хорошо иметь возможность управлять светодиодом из вашего веб-браузера, но мы действительно хотим иметь возможность управлять им из красивого графического интерфейса из любой точки мира.

Это именно то, что мы собираемся сделать в этом разделе, используя службу, называемую приборной панелью aREST. Фактически мы будем использовать ее не только для управления светодиодом, но и для изменения яркости светодиода с помощью ползунка прямо из вашего браузера.

Если это еще не сделано, создайте учетную запись по адресу:

```
http://dashboard.arest.io/
```

Вы должны иметь возможность создать новую панель управления из основного интерфейса:



Теперь внутри приборной панели мы собираемся создать новый элемент для управления и затемнения светодиода.

Создайте элемент панели управления типа Analog, а также вставьте идентификатор устройства, которым вы хотите управлять. Также не забудьте установить пин на 5:

Вы должны сразу увидеть вновь созданный элемент на панели инструментов:

Теперь попробуйте только что созданный слайдер в действии:

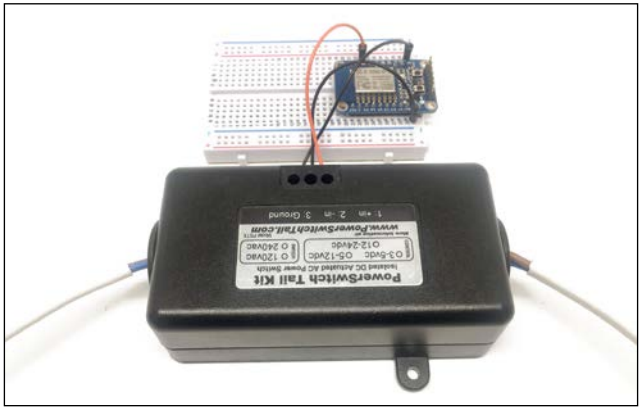
Вы должны заметить, что всякий раз, когда вы перемещаете этот ползунок и отпускаете мышь, он должен соответственно уменьшать яркость светодиода. Это, например, идеально подходит для затемнения светодиодной лампы в вашем доме.

Управление лампой из любой точки мира

Теперь мы собираемся использовать те же принципы и код, которые мы использовали до сих пор для другого приложения: управления лампой (или любым электрическим прибором) из любой точки мира.

Конфигурация этого раздела очень проста: сначала поместите модуль ESP8266 на макетную плату. Затем подключите контакт V_{in} + хвостовика PowerSwitch к контакту 5 ESP8266. Наконец, подключите два оставшихся контакта PowerSwitch Tail к контактам GND ESP8266.

Это изображение окончательного результата:

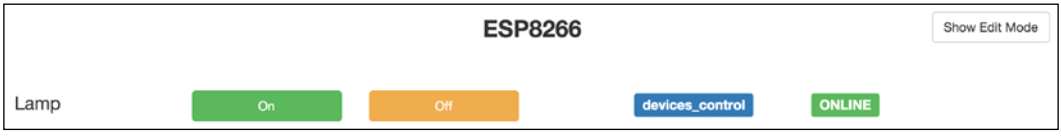


Затем подключите лампу (или другой электроприбор по вашему выбору) к хвостовику PowerSwitch, а другой конец PowerSwitch - к электросети.

Для кода просто используйте тот же код, что и в предыдущем разделе: основные изменения произойдут только внутри самой панели.

Внутри панели инструментов удалите предыдущий элемент слайдера и создайте новый типа Digital, используя те же параметры, что и раньше:

Вы должны увидеть, что вновь созданный элемент сразу появится внутри панели:



Теперь вы можете продолжить и протестировать эти кнопки: вы должны заметить, что они немедленно активируют или деактивируют лампу или устройство, подключенное к PowerSwitch.

В качестве иллюстрации проекта в реальной жизни, вот изображение только что созданного устройства, подключенного к настольной лампе:

Теперь вы можете управлять любым электрическим устройством из любой точки мира, используя облачную панель управления, которую мы использовали для этого проекта.

Резюме

Подведем итоги того, чего мы достигли в этом проекте. Мы использовали протокол MQTT для управления светодиодом, лампой или любыми электрическими устройствами из любой точки мира через чип ESP8266 Wi-Fi. Мы настроили ESP8266 как MQTT-клиент, а затем подключились к панели управления aREST, чтобы мы могли управлять им из любой точки мира.

Вы можете многое сделать, чтобы улучшить этот проект. Вы можете, например, добавить больше модулей, подобных этому, на панель управления aREST, чтобы удаленно управлять всеми лампами в вашем доме. Вы также можете добавить на свою панель управления другие устройства, например датчики, что мы увидим позже в этой книге.

В следующей главе мы рассмотрим еще одну очень важную тему Интернета вещей: как заставить наши устройства ESP8266 взаимодействовать с веб-службами, такими как Facebook или Twitter.

5

Взаимодействие с веб-службами

До сих пор в этой книге мы видели, как контролировать и управлять нашим модулем Wi-Fi ESP8266 из любой точки мира. Однако это лишь небольшая часть того, что мы можем сделать в рамках Интернета вещей.

В этой главе мы будем использовать ESP8266 для взаимодействия с существующими веб-службами и, следовательно, заставить физический мир взаимодействовать с этими службами через ESP8266. В качестве примеров мы подключим чип к службе Yahoo Weather, а затем к Twitter и Facebook. Давайте начнем!

Аппаратные и программные требования

Для этого проекта вам, конечно же, понадобится микросхема ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266.

Мы также будем публиковать данные в Твиттере через ESP8266. Для этого я буду использовать простой датчик температуры и влажности DHT11. Конечно, для этой задачи вы легко можете использовать другой датчик.

Наконец, вам также понадобятся перемычки и макетная плата.

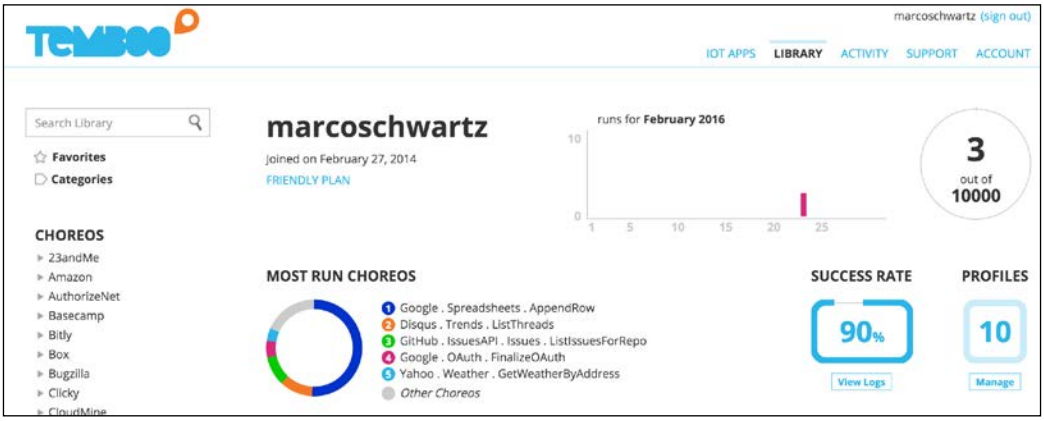
Это список всех компонентов, которые будут использоваться в этом проекте:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- Датчик DHT11
- Макетная плата
- Перемычки

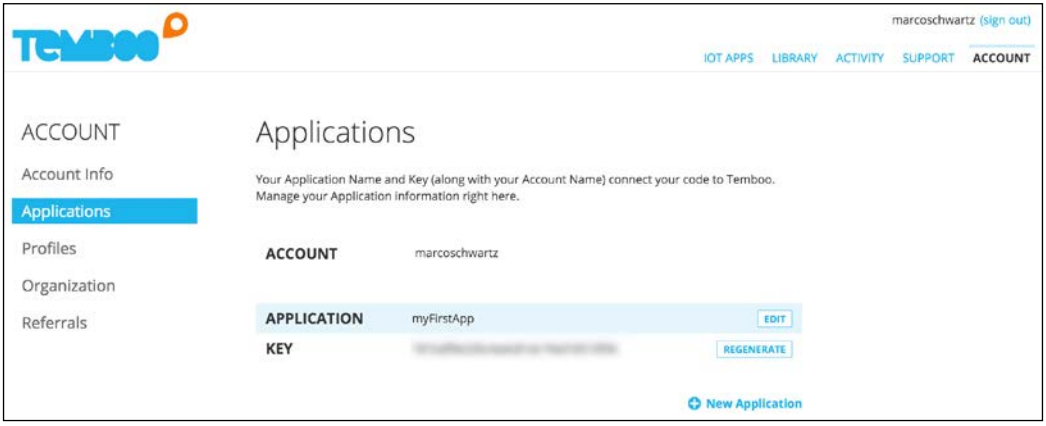
Что касается программы, вам нужно будет установить последнюю версию Arduino IDE, которую вы можете получить отсюда:

```
http://www.arduino.cc/en/Main/Software
```

Вам также необходимо будет создать аккаунт на веб-сайте Temboo. Мы будем использовать Temboo для взаимодействия ESP8266 с веб-службами, такими как Yahoo или Twitter.



После этого перейдите в Account и создайте первое приложение. Вам понадобится имя приложения и ключ позже в этой главе:



Вам также понадобится специальная версия библиотеки Temboo, которую я модифицировал для работы с ESP8266. Вы можете найти библиотеку в папке с кодами.

Затем перейдите в папку основной библиотеки Arduino (для Windows в папке C: \ Program Files (x86) \ Arduino \ libraries; для OS X нажмите «Показать содержимое пакета» в приложении Arduino). Обязательно сохраните существующую библиотеку Temboo, чтобы вы могли повторно использовать ее позже для других проектов, кроме ESP9266. После этого удалите существующую библиотеку Temboo и замените ее той, которую вы получили из кода для этой главы.

Получение данных о погоде из Yahoo

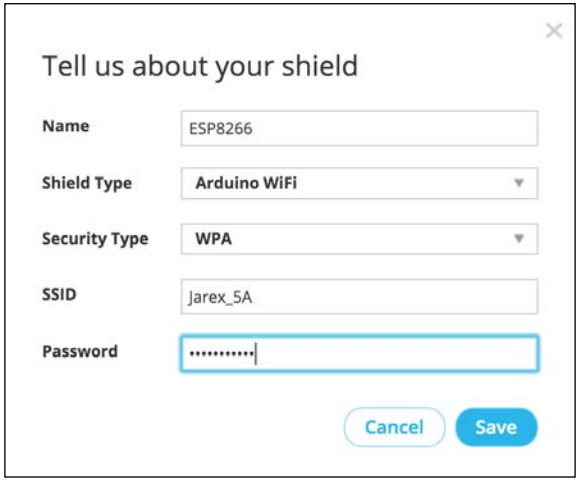
Для первого проекта в этой главе мы фактически узнаем, как получать данные о погоде из службы погоды Yahoo. Мы увидим, что это действительно просто благодаря Temboo:

1. Сначала перейдите по следующему URL-адресу:

   ```
   https://temboo.com/library/Library/Yahoo/Weather/
   GetWeatherByAddress/
   ```

2. Поскольку вы впервые используете Temboo, вам нужно будет создать новый «шилд» для службы Temboo. Не волнуйтесь, я знаю, что мы здесь не используем шилд Arduino; на момент написания этой книги Temboo просто не поддерживал чип ESP8266 Wi-Fi.

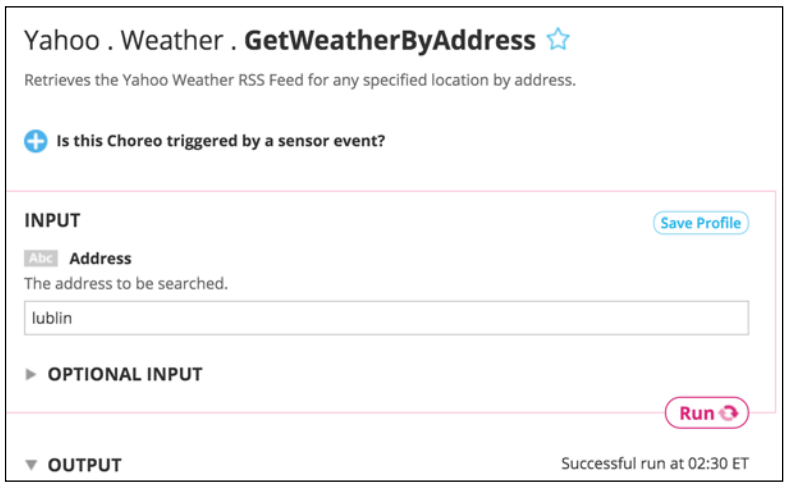
   Лучшее, что мы можем сделать, - это выбрать шилд Wi-Fi для Arduino, который очень близок с точки зрения программного обеспечения. На этом же экране также введите SSID и пароль вашей сети Wi-Fi:



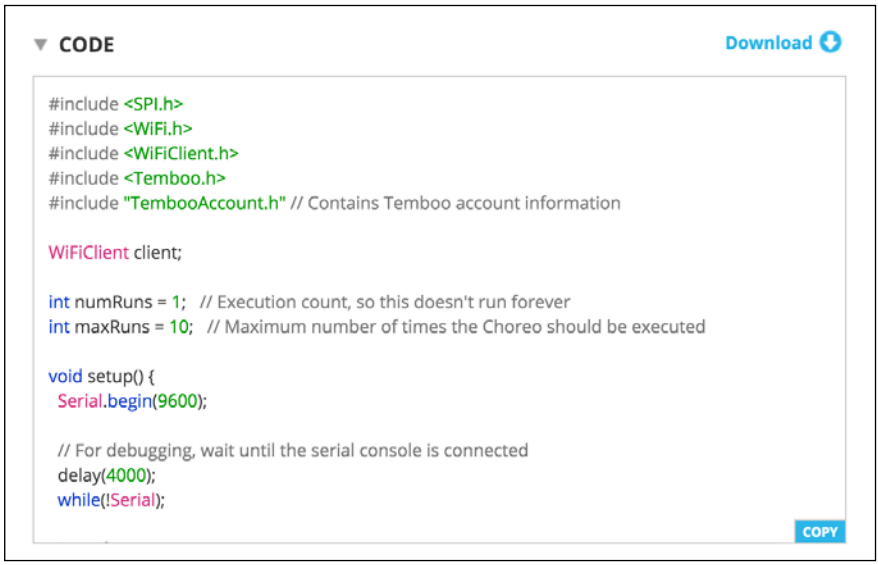
Это позволит вам не вводить эти параметры каждый раз заново. Теперь вы должны увидеть свой недавно созданный шилд на той же странице:

Теперь мы посмотрим, как автоматически сгенерировать код для нашего проекта. Тогда нам просто нужно будет немного изменить его для наших нужд.

3. Теперь укажите свой город в интерфейсе Temboo:



4. Затем вы можете протестировать его, нажав кнопку «Run -Выполнить», или сгенерировать код:



5. Теперь загрузите код и поместите его в папку на вашем компьютере.Вы можете использовать код из папки с кодами.
6. Теперь мы рассмотрим детали этого кода. На самом деле вам нужно изменить только две строчки, если вы хотите использовать этот скетч с ESP8266. Вам необходимо изменить клиентские библиотеки Wi-Fi и Wi-Fi для библиотеки ESP8266WiFi:

```
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h" // Contains Temboo account information
```

Затем мы определяем, сколько раз мы хотим запустить этот код (Temboo оплачивается после нескольких вызовов):

```
int numRuns = 1;   // Execution count, so this doesn't run forever
int maxRuns = 10;   // Maximum number of times the Choreo should
be executed
```

7. Внутри функции скетча setup () подключаем плату к сети Wi-Fi:

```
  int wifiStatus = WL_IDLE_STATUS;

  // Установите, присутствует ли WiFi шилд
  Serial.print("\n\nShield:");
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("FAIL");

    // Если нет WiFi-шилда, остановитесь здесь
    while(true);
  }

  Serial.println("OK");

  // Попробуйте подключиться к локальной сети Wi-Fi
  while(wifiStatus != WL_CONNECTED) {
    Serial.print("WiFi:");
    wifiStatus = WiFi.begin(WIFI_SSID, WPA_PASSWORD);

    if (wifiStatus == WL_CONNECTED) {
      Serial.println("OK");
    } else {
      Serial.println("FAIL");
    }
    delay(5000);
  }

  Serial.println("Setup complete.\n");
}
```

8. Затем в функции скетча loop () мы отправляем запрос в Temboo и печатаем результат внутри последовательного монитора:

```
  if (numRuns <= maxRuns) {
    Serial.println("Running GetWeatherByAddress - Run #" +
String(numRuns++));

    TembooChoreo GetWeatherByAddressChoreo(client);
```

```
    // Вызов клиента Temboo
    GetWeatherByAddressChoreo.begin();

    // Установить аккаунт Temboo
    GetWeatherByAddressChoreo.setAccountName(TEMBOO_ACCOUNT);
    GetWeatherByAddressChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
    GetWeatherByAddressChoreo.setAppKey(TEMBOO_APP_KEY);

    // Установить входы Choreo
    String AddressValue = "lublin";
    GetWeatherByAddressChoreo.addInput("Address", AddressValue);

    // Определите Choreo для запуска
    GetWeatherByAddressChoreo.setChoreo("/Library/Yahoo/Weather/
GetWeatherByAddress");

        // Запустите Choreo; когда результаты будут доступны,
распечатайте их в серийный номер
    GetWeatherByAddressChoreo.run();

    while(GetWeatherByAddressChoreo.available()) {
      char c = GetWeatherByAddressChoreo.read();
      Serial.print(c);
    }
    GetWeatherByAddressChoreo.close();
  }

  Serial.println("\nWaiting...\n");
  delay(30000); // wait 30 seconds between GetWeatherByAddress
calls
```

9. Пришло время протестировать этот первый проект главы! Просто проверьте внутри Temboo.h, что все параметры верны, переведите плату в режим загрузчика (чтобы ее можно было запрограммировать), а затем загрузите код на плату.

10. После этого откройте последовательный монитор. Вы должны увидеть, что плата получает данные из службы погоды Yahoo и отображает их на последовательном мониторе:

Вы можете проверить местную температуру, влажность и другие данные о погоде прямо со своего ESP8266!

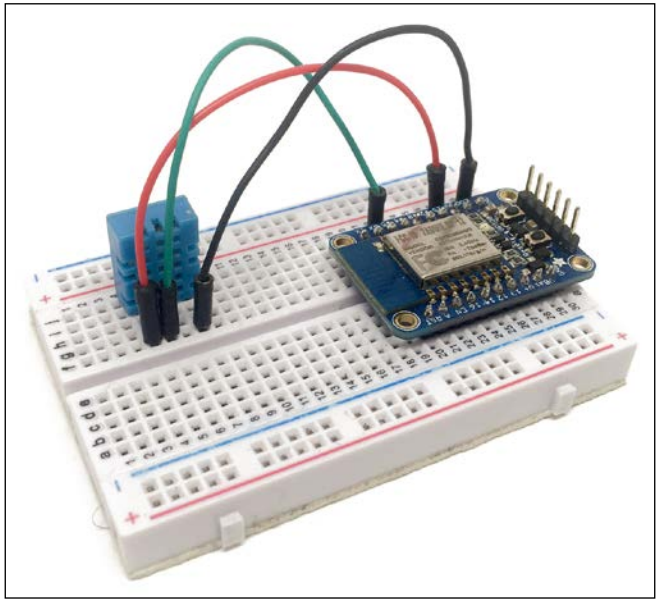
Публикация данных о температуре и влажности в Twitter

В этом втором проекте главы мы увидим, как на самом деле публиковать измеренные данные в вашем аккаунте Twitter.

Сначала нам нужно настроить модуль ESP8266, что означает создание проекта с датчиком DHT11.

Просто поместите плату ESP8266 на макетную плату, а затем поместите датчик DHT11 рядом с ней. Подключите первый вывод датчика DHT11 к выводу VCC ESP8266, второй вывод к выводу 5 платы ESP и, наконец, последний вывод датчика к одному выводу GND платы ESP.

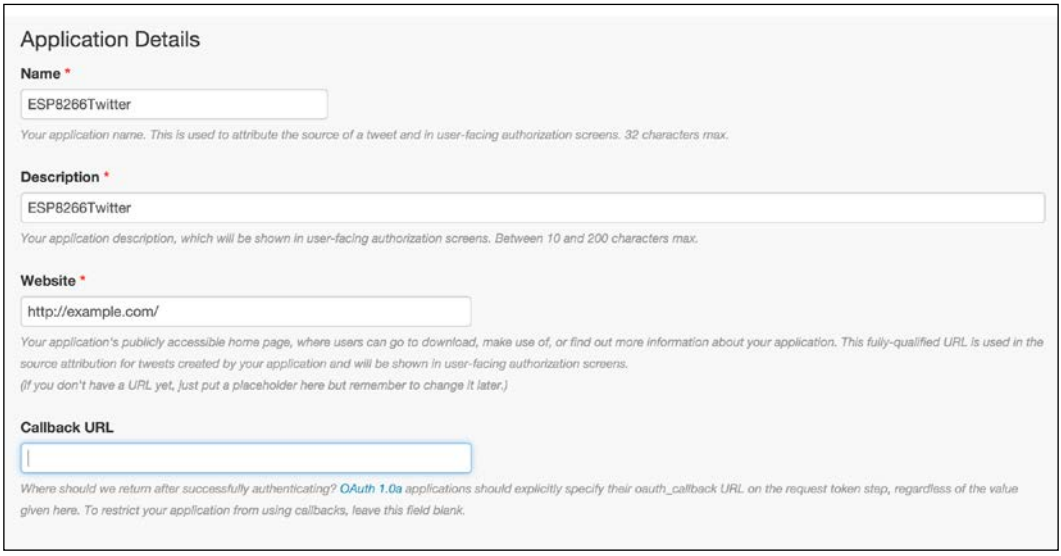
Это окончательный результат:



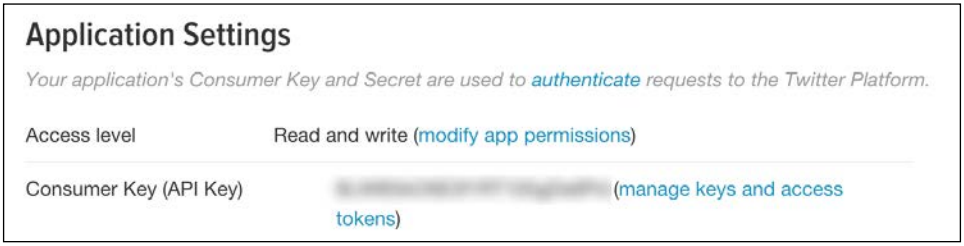
1. Теперь нам нужно зарегистрировать приложение в Twitter, прежде чем мы сможем создавать новые твиты с нашими данными. Для этого перейдите по ссылке:
   `https://apps.twitter.com/`

   Вам будет предложено войти в свою аккаунт Twitter. Затем вы можете увидеть свои существующие приложения, если они у вас есть:

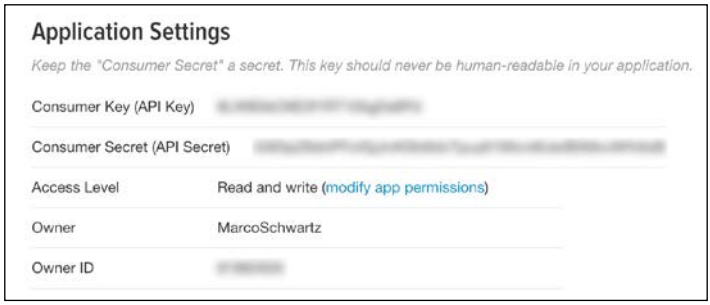
2.Нажмите кнопкуCreateNew Appи дайте ему имя и URL-адрес по умолчанию (это не имеет значения):



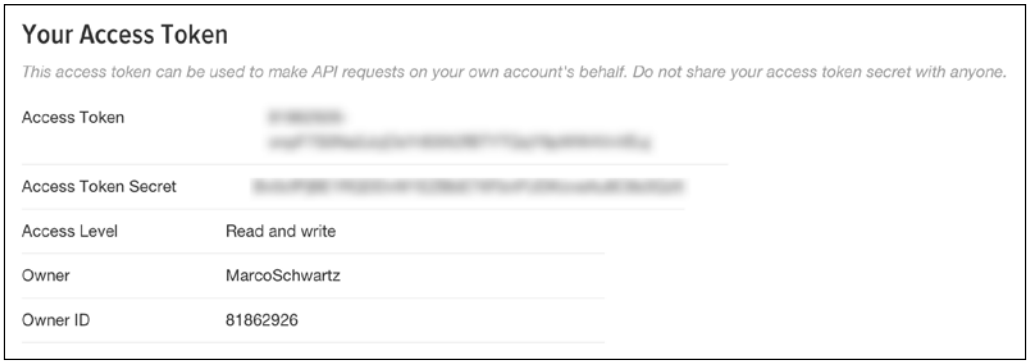
После его создания вы сможете получить доступ к важным параметрам, таким как ключ потребителя:



На вкладке настроек вы также сможете увидеть секретный ключ API:



3. Затем на той же странице создайте токен для приложения:

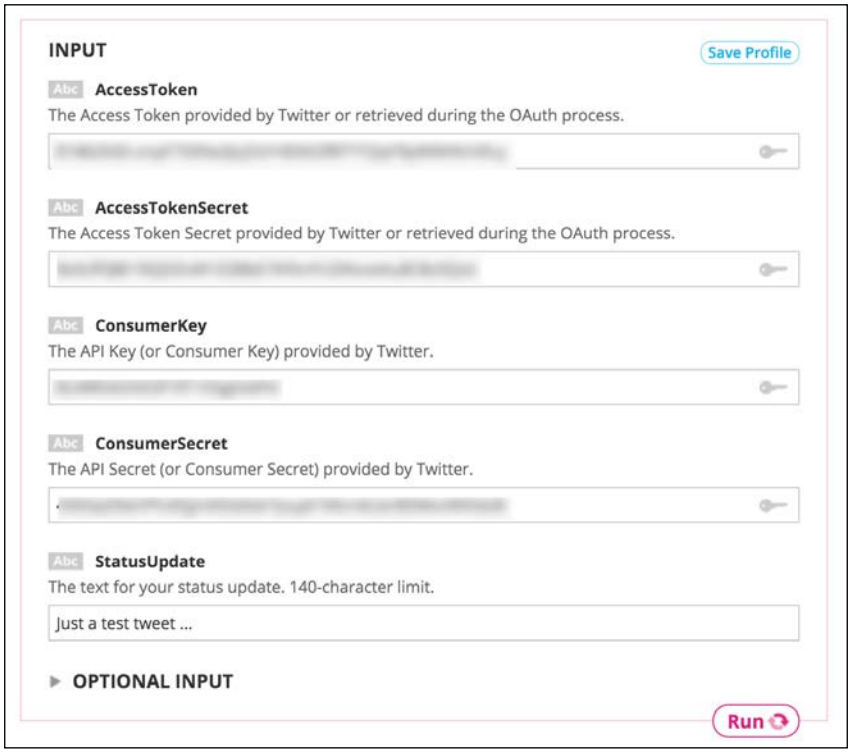


Это даст вам токен доступа и секретный токен. Эти четыре вам потребуются для настройки проекта в Тембу.

4. Теперь перейдите к:

```
https://temboo.com/library/Library/Twitter/Tweets/
StatusesUpdate/
```

Вы сможете заполнить четыре параметра из приложения Twitter:



5. После этого снова сгенерируйте код для скетча. Не беспокойтесь о содержании обновления статуса; мы изменим это позже.
6. Как и раньше, вам просто нужно немного изменить скетч, поэтому включите библиотеку WiFi ESP8266 и вставьте библиотеку DHT:

```
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h" // Содержит информацию об аккаунте Temboo
#include "DHT.h"
```

7. После этого вставляем код датчика DHT11:

```
// Выводы датчика DHT11
#define DHTPIN 5#define DHTTYPE DHT11


// Инициализируйте датчик DHT с соответствующ. опцией для ESP8266
DHT dht(DHTPIN, DHTTYPE, 15);
```

8. Внутри функции setup () нам нужно инициализировать датчик DHT11:

```
// Init DHT
dht.begin();
```

9. Затем внутри функции loop () нам нужно добавить только код для чтения данных с датчика:

```
// Read data
float h = dht.readHumidity();
float t = dht.readTemperature();
```

10. Нам также необходимо изменить значение строки обновления статуса, чтобы вставить измерения, сделанные датчиком:

```
String StatusUpdateValue = "The temperature is " + String(t) + "
and the humidity is " + String(h) + ".";
```

11. Пришло время протестировать скетч! Проверьте параметры в файле Temboo.h и после этого загрузите код на плату ESP8266.

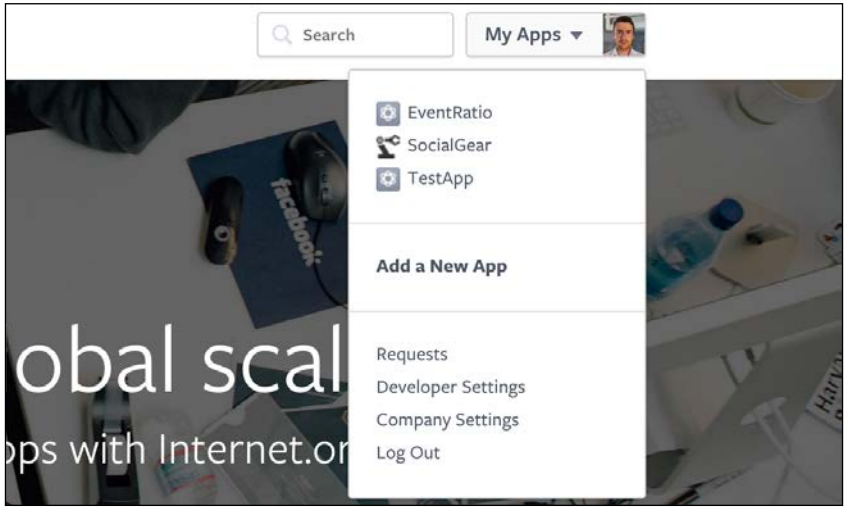
Через некоторое время вы должны увидеть, что в вашем аккаунте был опубликован новый твит вместе с данными с датчика:

Поздравляем, теперь вы можете публиковать данные со своего ESP8266 в аккаунте Twitter!

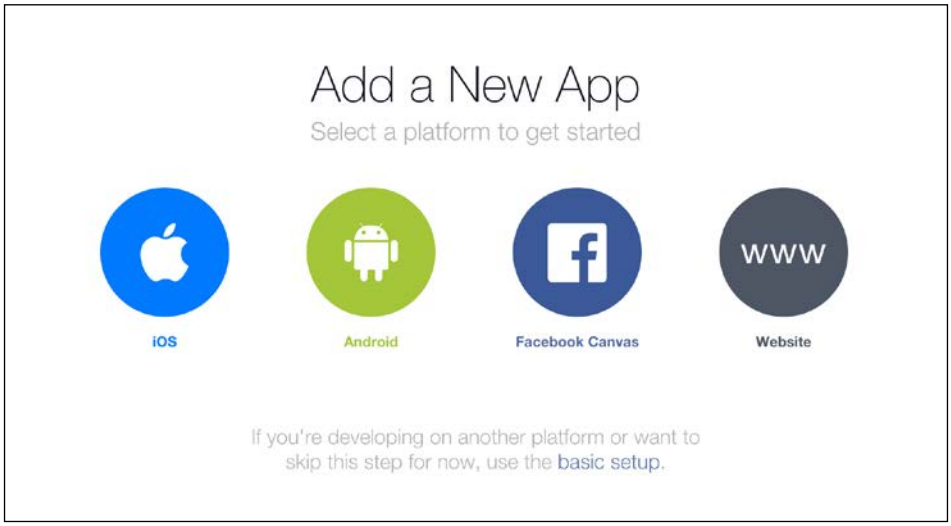
Создание нового сообщения в Facebook из ESP8266

В последнем проекте этой главы мы увидим, как взаимодействовать с Facebook с Wi-Fi-чипа ESP8266 через Temboo. Мы увидим, как просто опубликовать обновление статуса, но вы можете использовать его для публикации чего-либо на стене друга, публикации данных на странице и многого другого!

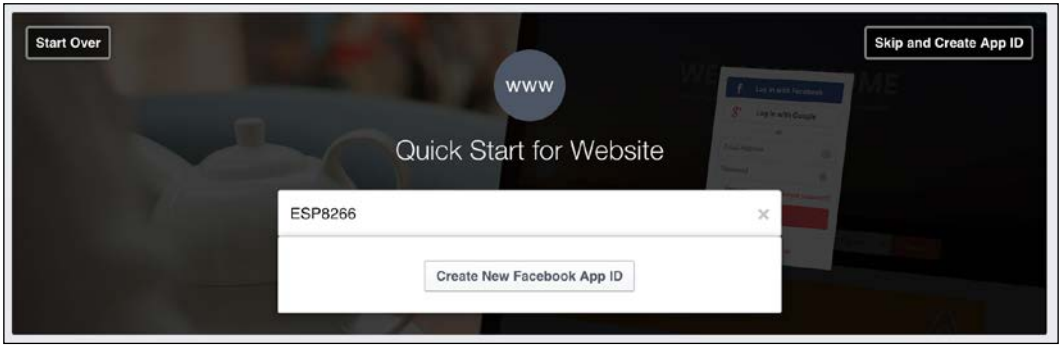
1. Первый шаг - создать приложение Facebook. Вы можете сделать это, перейдя по ссылке:
   `https://developers.facebook.com/`
2. Оттуда нажмите на **Add a New App**:



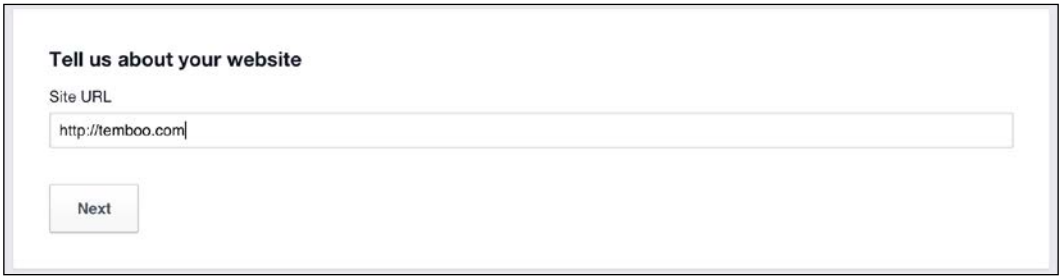
3. Когда интерфейс запросит у вас тип приложения, выберите Website:



4. Затем дайте своему приложению имя:

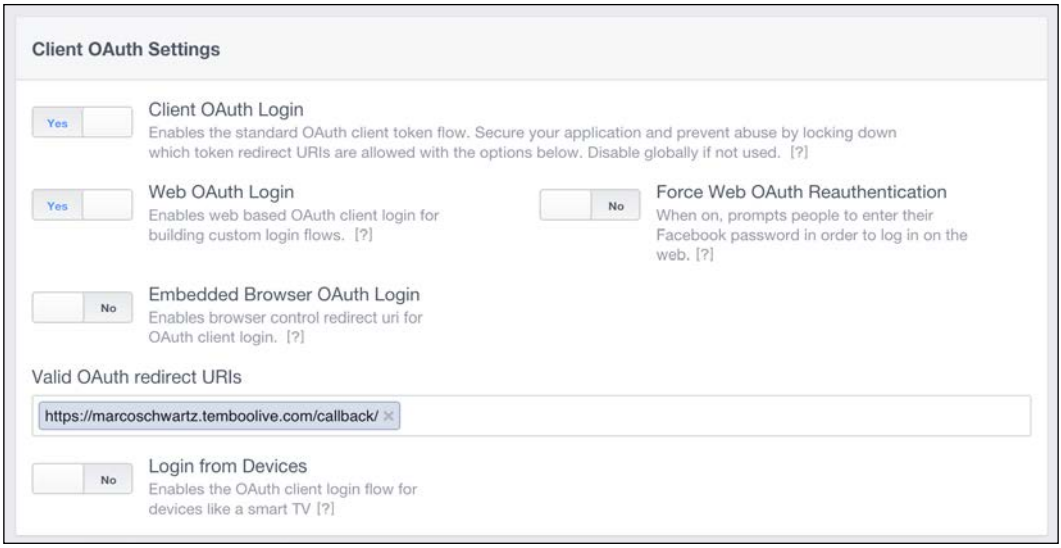


5. Затем вас попросят указать URL-адрес. Укажите произвольный адрес (это не будет использоваться проектом),где вы можете поместить все, что угодно:



6. После этого шага ваше приложение будет создано. Что вам нужно,чтобы получить здесь ID приложения и секретное приложение, которы должны появиться на панели инструментов приложения:

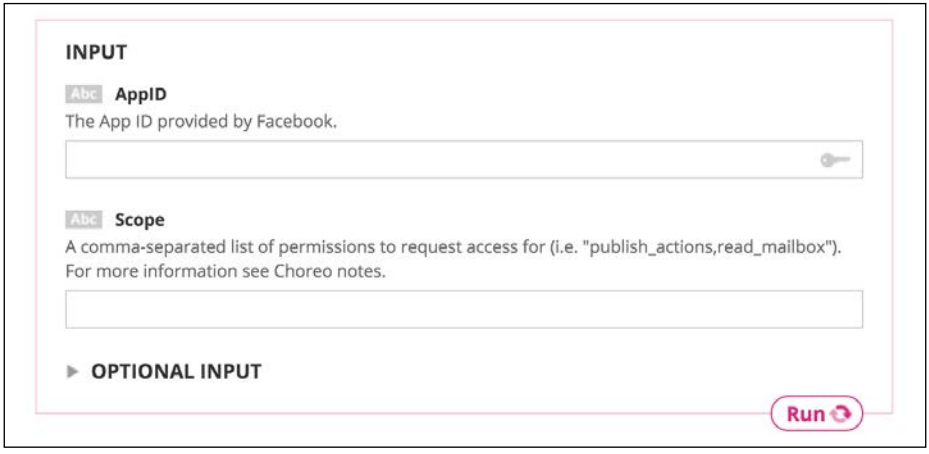
77. Затем перейдите в раздел«Setting - Настройки» и найдите «Client OAuth Settings - Настройки клиента OAuth». Добавьте URL-адрес, указанный на этом снимке экрана,в качестве URL-адреса обратного вызова заменив имя пользователя Temboo:



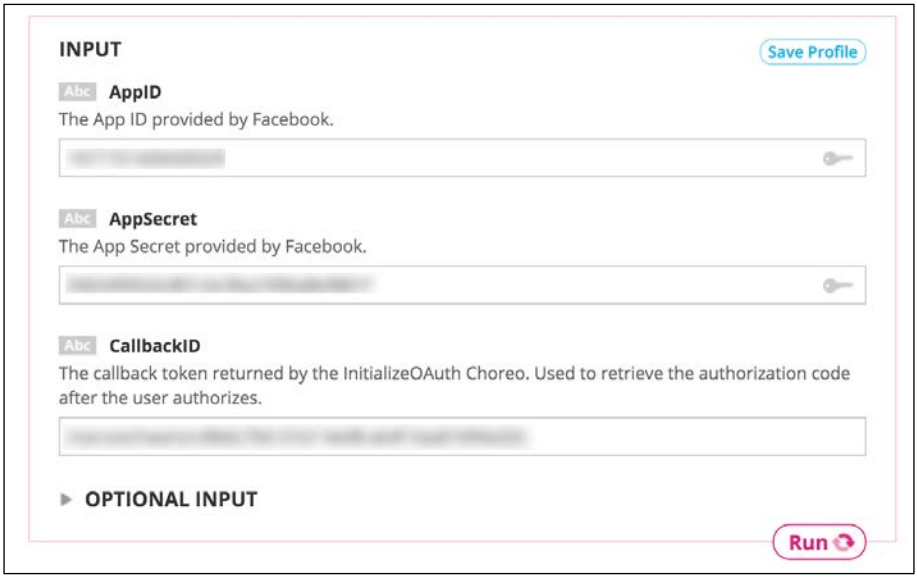
8. Вы настроены на позицию Facebook! Теперь вернемся к Temboo:

   `https://www.temboo.com/library/Library/Facebook/OAuth/`

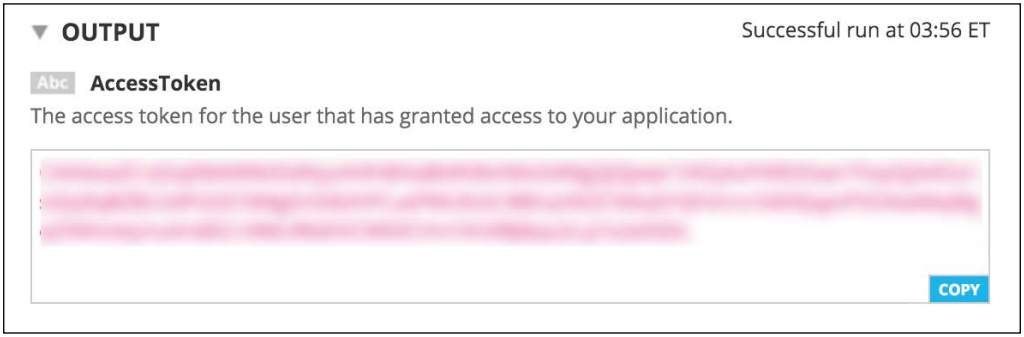
9. Эта страница просто позволит вам получить токен доступа Facebook. Сначала введите идентификатор приложения (который вы получили ранее на веб-сайте Facebook) и область действия (используйте publish_actions)



10.После этого вам будет предложено перейти по ссылке. Сделайте это, авторизуйте приложение, а затем вернитесь на страницу окончательной авторизации и введите все необходимые данные:



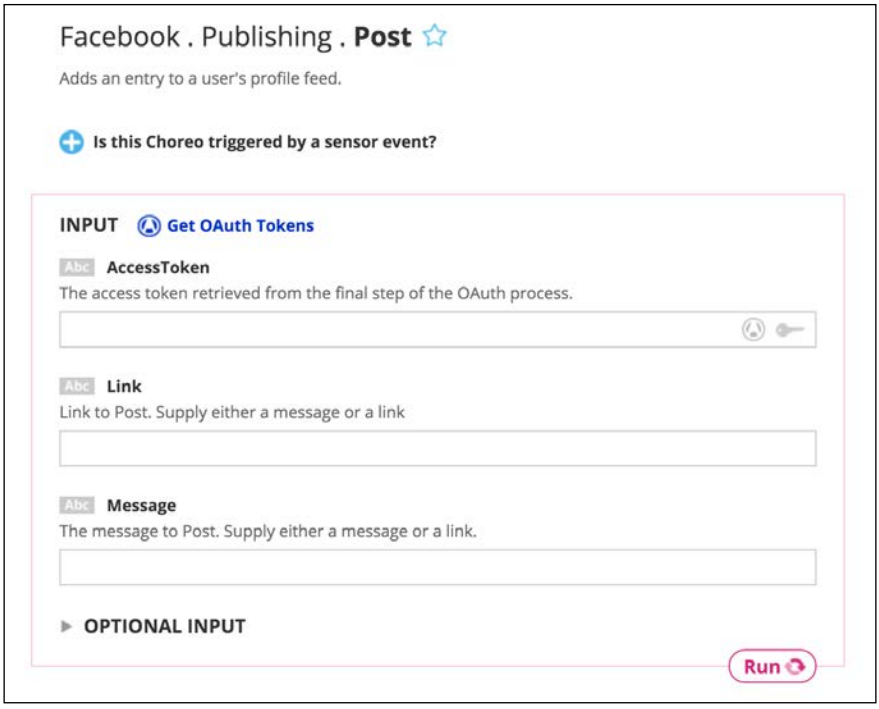
11. После нажатия кнопки « Run - Выполнить» вы получите токен доступа:



12. Теперь перейдите в библиотеку Temboo, которую мы действительно хотим использовать, то есть библиотеку для публикации поста на Facebook:

```
https://temboo.com/library/Library/Facebook/Publishing/Post/
```

13. Оттуда введите свой токен доступа, а также сообщение, которое вы хотите, чтобы ESP8266 разместил на своей стене:



Вот как это должно выглядеть в конце:

14. Затем вы можете сгенерировать код и загрузить его. Как и раньше, нам нужно изменить внутри кода только две вещи. Единственное, что вам нужно здесь изменить, это библиотеку Wi-Fi:

```
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h" // Содержит информацию об аккаунте Temboo
```

15. Затем внутри функции loop () мы размещаем сообщение на стене Facebook на каждой итерации цикла:

```
if (numRuns <= maxRuns) {
    Serial.println("Running Post - Run #" + String(numRuns++));

    TembooChoreo PostChoreo(client);

    // Вызов клиента Temboo
    PostChoreo.begin();

    // Установить учетные данные Temboo
    PostChoreo.setAccountName(TEMBOO_ACCOUNT);
    PostChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
    PostChoreo.setAppKey(TEMBOO_APP_KEY);

    // Установить входы Choreo
    String MessageValue = "A simple message from the ESP8266!";
    PostChoreo.addInput("Message", MessageValue);
    String AccessTokenValue = "accessToken";
    PostChoreo.addInput("AccessToken", AccessTokenValue);

    // Определите Choreo для запуска
    PostChoreo.setChoreo("/Library/Facebook/Publishing/Post");

    //Запустите Choreo; когда результаты будут доступны, распечатайте их
    //   в серийный номер


    PostChoreo.run();

    while(PostChoreo.available()) {
      char c = PostChoreo.read();
      Serial.print(c);
    }
    PostChoreo.close();
  }
```

```
    Serial.println("\nWaiting...\n");
    delay(30000); // wait 30 seconds between Post calls
}
```

16. Пришло время наконец-то протестировать этот проект! Загрузите код на плату, а затем проверьте свой профиль в Facebook. Через некоторое время вы должны увидеть новую запись, появившуюся на вашей стене, с сообщением, которое мы определили в скетче:

Теперь вы можете использовать Temboo для публикации сообщений в Facebook с помощью Wi-Fi-чипа ESP8266!

Резюме

В этой главе мы узнали, как взаимодействовать с веб-сервисами с помощью микросхемы ESP8266.

Мы узнали, как публиковать данные в Twitter, как получать данные о погоде из Yahoo, а также как взаимодействовать с Facebook.

Я действительно рекомендую вам сейчас поиграть со всеми библиотеками, которые предлагает Temboo.

Возможности практически безграничны. Например, у вас может быть страница Faceboook, которая предназначена только для вашего дома и которую вы используете для автоматической публикации данных о своем доме! Вы также можете просто использовать дополнительные данные, предоставляемые этими сервисами (например, данные о погоде), для обогащения ваших собственных проектов.

В следующей главе мы узнаем о другой важной части Интернета вещей: межмашинных коммуникациях.

6

Межмашинная связь

В предыдущих главах мы узнали, как сделать данные журнала ESP8266 онлайн, а также как управлять ими удаленно из любой точки мира. Однако у всех этих проектов Интернета вещей было что-то общее: в какой-то момент они требовали вашего вмешательства - либо для просмотра данных, либо для того, чтобы нажимать кнопки для удаленного управления устройством.
В этой главе мы рассмотрим еще одну область Интернета вещей: межмашинную связь (M2M). Это те случаи, когда не требуется вмешательства человека, и когда машины напрямую взаимодействуют друг с другом для выполнения определенных задач.
И это именно то, что мы собираемся проиллюстрировать на платах ESP8266 в этой главе. Сначала мы сделаем очень простой проект, чтобы проиллюстрировать связь M2M, а затем применим эти знания для создания светового реле на основе двух плат ESP8266. Давайте начнем!

Аппаратные и программные требования

Для этого проекта вам, конечно же, понадобится микросхема ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266.

Для первого проекта этой главы вам понадобится светодиод и резистор на 330 Ом. Вам также понадобится мини-кнопка и резистор на 1 кОм.

Для второго проекта этой главы вам понадобится реле, резистор 10 кОм и небольшой фотоэлемент.

Вам понадобится USB-модуль FTDI 3,3 В / 5 В для программирования микросхемы ESP8266.
Наконец, вам также понадобятся перемычки и макетная плата.

Это список всех компонентов, которые будут использоваться в этой главе:

- Adafruit ES8266 модуль (x2)
- USB-модуль FTDI
- LED
- Резистор 330 Ом
- Реле
- Фотоэлемент
- Резистор 10 кОм
- Кнопка
- Резистор 1 кОм
- Макетная плата (x2)
- Перемычки

Что касается программы, установите последнюю версию Arduino IDE, которую вы можете получить по адресу:

```
http://www.arduino.cc/en/Main/Software
```

Затем установите библиотеки aREST и PubSubClient,используя менеджер библиотек Arduino.

Чтобы чипы ESP8266 взаимодействовали друг с другом, мы также собираемся использовать веб-сервис IFTTT. IFTTT - это служба, в которой вы можете создавать рецепты, которые будут выполнять заданное действие при получении заданного запуска, что идеально подходит для связи M2M. Если это еще не сделано, создайте аккаунт на IFTTT:

```
https://ifttt.com/
```

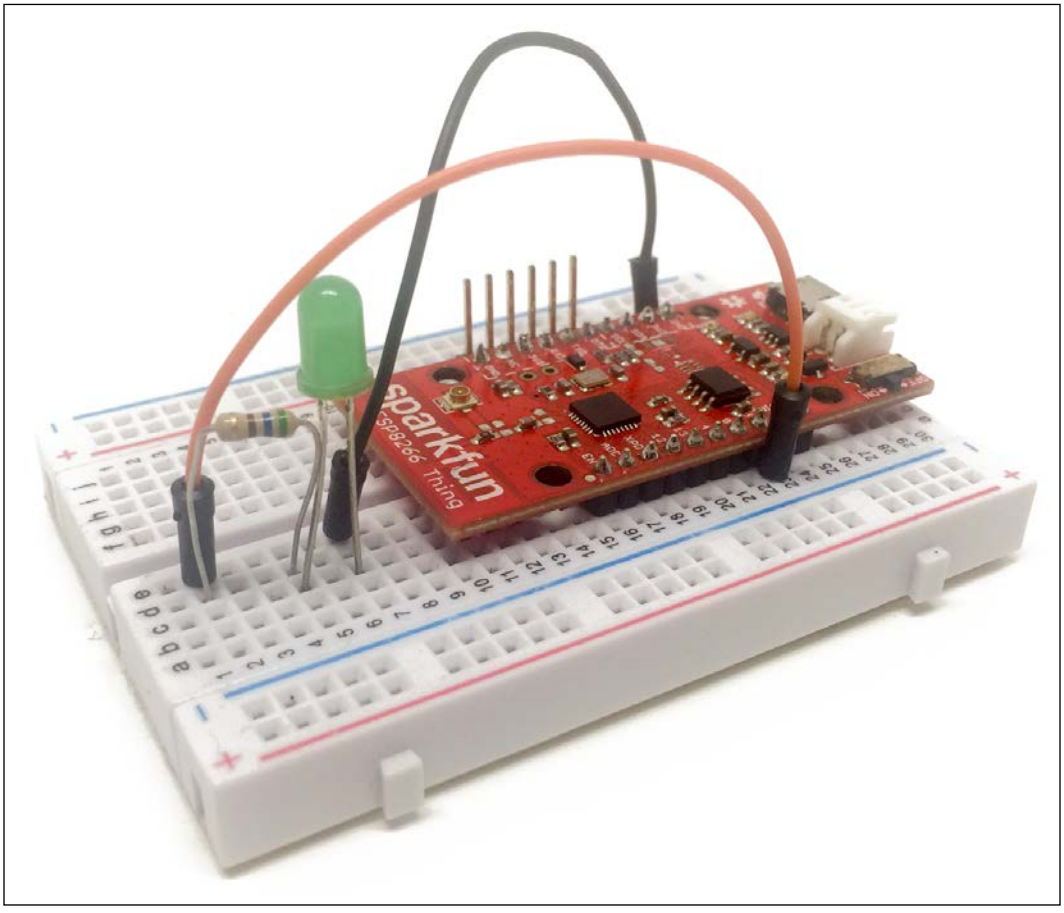
Там вы уже можете добавить канал Maker в свою учетную запись. Это гарантирует, что он доступен для ваших рецептов, и вы также получите ключ, который вам понадобится позже в этой главе.

Простая межмашинная связь

В первом проекте этой главы мы рассмотрим очень простой случай M2M-связи, когда один чип Wi-Fi ESP8266 отправит сигнал запуска другому чипу (конечно, через облако), который в ответ переключится состояние светодиода. В качестве запуска действия мы будем использовать простую кнопку, подключенную к первому ESP8266.

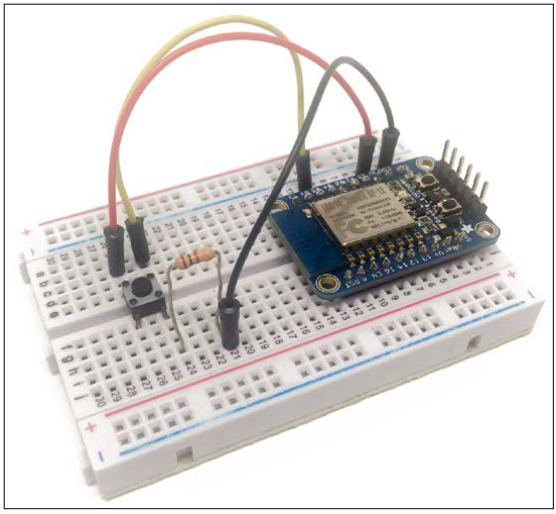
Сначала подберем компоненты для этого проекта. Для платы светодиодов аппаратная конфигурация действительно проста: просто подключите резистор последовательно со светодиодом, как мы уже делали в предыдущих главах. Затем подключите резистор к выводу 5 ESP8266, а другой вывод к земле.

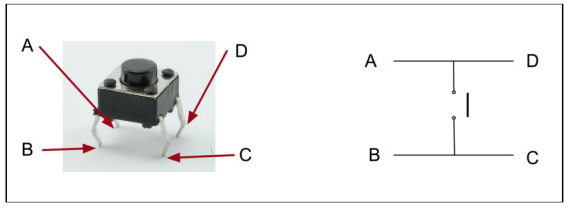
Это окончательный результат для светодиодной платы:



На другой плате поместите кнопку и ESP8266. Подключите кнопку,резистор и перемычки согласно рисунку. Выводы 5 и V_{CC} чипа ESP8266 Wi-Fi подключите к кнопке. Наконец, подключите резистор к 5 выводу чипа,а другой вывод резистора - на землю.

Это окончательный результат для платы с кнопкой:





Теперь мы собираемся настроить обе платы, чтобы они могли взаимодействовать друг с другом. Как я уже сказал ранее, мы собираемся использовать сервис IFTTT, чтобы две платы общались друг с другом.

Сначала мы настроим светодиодную плату, чтобы она могла получать команды из облака. Для этого мы снова воспользуемся фреймворком aREST:

1. Этот скетч начинается с импорта необходимых библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

2. Далее мы создаем необходимых клиентов для подключения к облаку:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

3. Также инициализируем библиотеку aREST:

```
aREST rest = aREST(client);
```

4. Затем мы даем устройству уникальный идентификатор:

```
char* device_id = "01e47f";
```

Вам также необходимо ввести здесь учетные данные своей сети Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-password";
```

5. Затем мы определяем логическую переменную для хранения текущего состояния светодиода:

```
bool ledState;
```

Мы также определяем функцию переключения светодиода, которую мы реализуем позже:

```
int toggle(String command);
```

6. В функции скетча setup () выставляем функцию переключения на aREST, а также подключаем чип к сети Wi-Fi:

```
void setup(void)
{

  // Start Serial
  Serial.begin(115200);

  // Установить обратный звонок
  client.setCallback(callback);

  // Дайте имя и ID устройству
  rest.set_id(device_id);
  rest.set_name("led");


  // Функция
  rest.function("toggle", toggle);

  // Состояние светодиода
  ledState = false;
  // Подключиться к WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("");
  Serial.println("WiFi connected");

  // Pin 5 как выход
  pinMode(5, OUTPUT);

}
```

7. В функции скетча loop () мы просто сохраняем соединение с облачной платформой aREST:

```
void loop() {

  // Подключитесь к облаку
  rest.handle(client);

}
```

8. В конце нам также нужно реализовать функцию переключения светодиода. Мы просто инвертируем состояние светодиода при вызове функции, а затем применяем это новое состояние к светодиоду:

```
// Переключать LED
int toggle(String command) {

  ledState = !ledState;
  digitalWrite(5, ledState);
  return 1;
}
```

Вы также можете получить этот код из папки с кодами

Затем измените учетные данные внутри кода и загрузите их на плату, на которой расположен светодиод.

Теперь мы посмотрим, как настроить плату, на которой расположена кнопка, которая будет запускать событие в IFTTT при каждом нажатии кнопки:

1. Это начинается с включения библиотеки Wi-Fi ESP8266:

```
#include <ESP8266WiFi.h>
```

2. Затем вам также необходимо определить свои учетные данные Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

3. Вам также необходимо вставить свой ключ IFTTT на этом этапе:

```
const char* host = "maker.ifttt.com";
const char* eventName = "button_pressed";
const char* key = "ifttt-key";
```

4. В функции setup () скетча мы просто подключаем ESP8266 к вашей сети Wi-Fi:

```
void setup() {
  Serial.begin(115200);
  delay(10);

  // Начнем с подключения к сети Wi-Fi

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Pin 5 as input
  pinMode(5, INPUT);
}
```

5. В функции скетча loop () проверяем, была ли нажата кнопка:

```
  if (digitalRead(5)) {
```

6. Если это так, мы отправляем сообщение в IFTTT. Сначала подключаемся к их серверам и создаем запрос:

```
  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
```

```
    Serial.println("connection failed");
    return;
  }

  // Теперь мы создаем URL для запроса
  String url = "/trigger/";
  url += eventName;
  url += "/with/key/";
  url += key;

  Serial.print("Requesting URL: ");
  Serial.println(url);
```

7. Как только это будет сделано, мы отправим запрос в IFTTT:

```
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  int timeout = millis() + 5000;
  while (client.available() == 0) {
    if (timeout - millis() < 0) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }
```

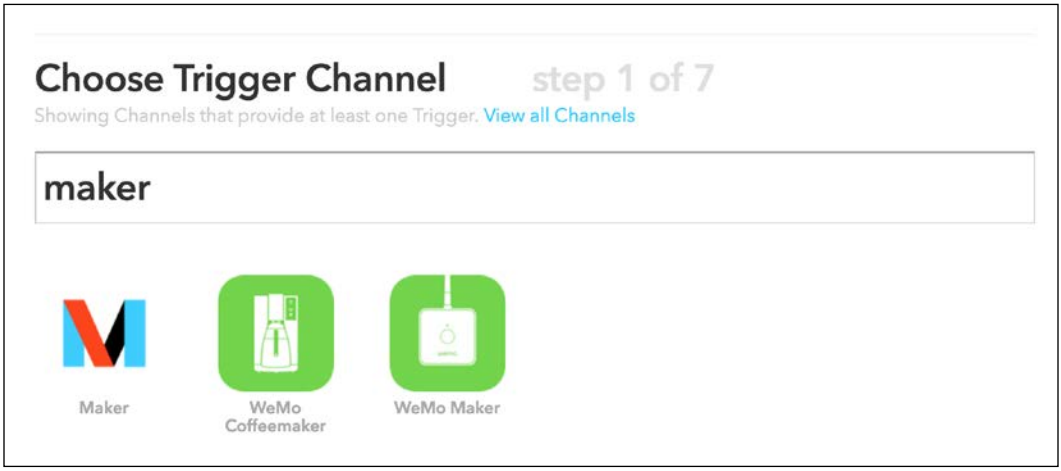
8. Чтобы завершить запрос, мы читаем данные, возвращаемые из IFTTT, и печатаем их:

```
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
}
```

9. Теперь вы также можете взять скетч из папки с кодами и загрузить его на плату.

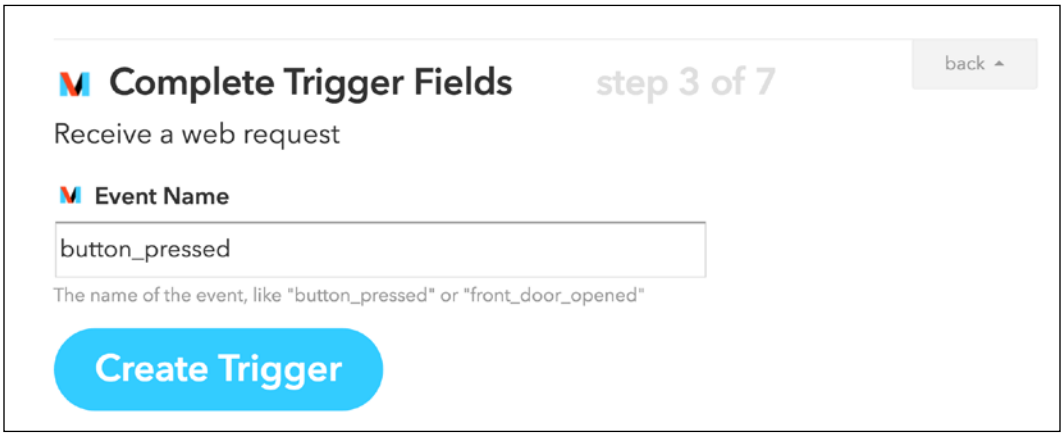
   Однако на данный момент две наши платы не взаимодействуют друг с другом. Вот почему мы собираемся перейти к IFTTT, чтобы создать рецепт для связи этих двух плат.

10. На веб-сайте IFTTT создайте новый рецепт и задействуйте канал Maker :

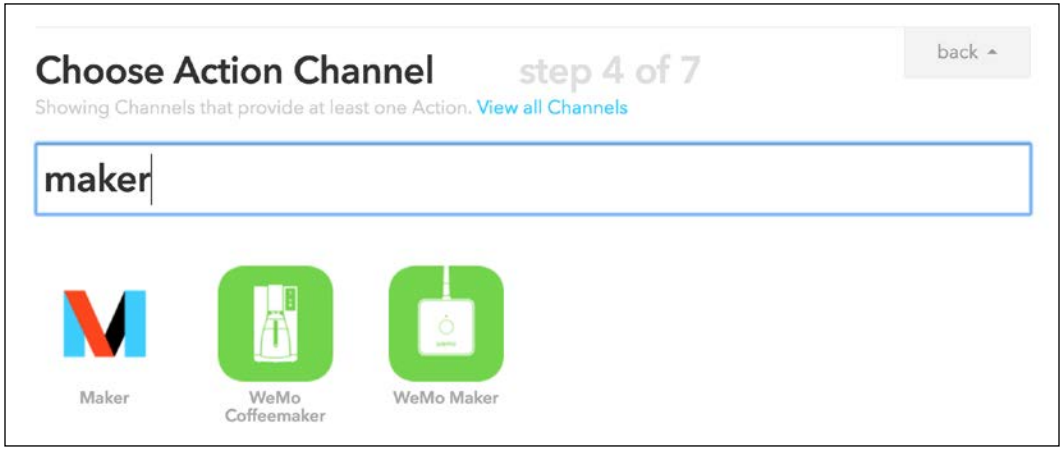


11. Затем в качестве типа запуска выберите Receive a web requestas - Получить веб-запрос:

12. В качестве события введите button_pressed, что мы и сделали внутри скетча Arduino:

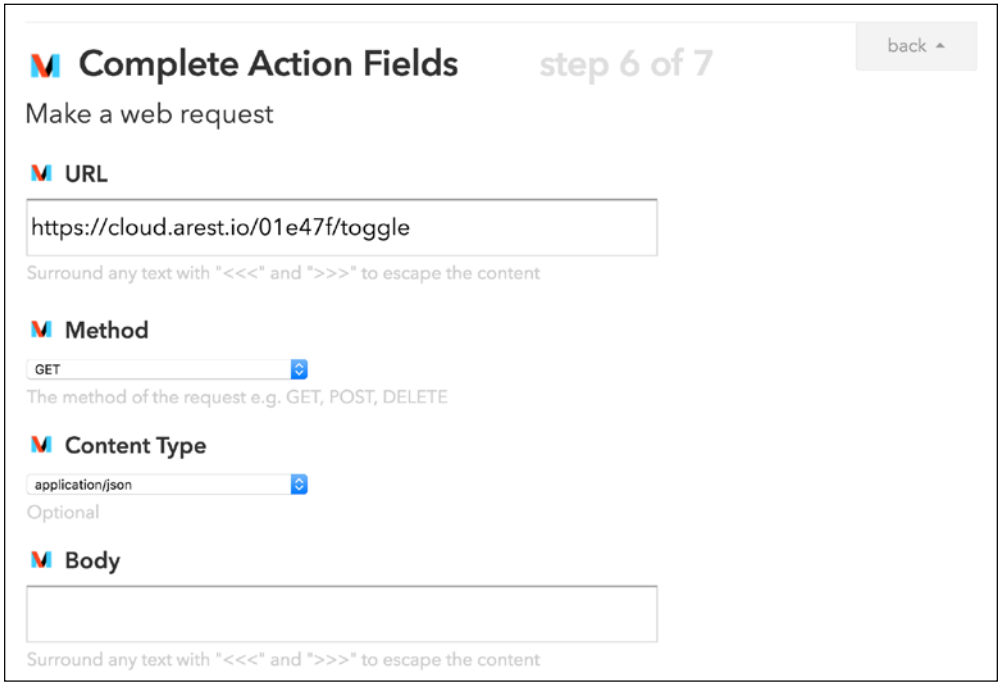


13. В качестве канала действия также выберите канал Maker:



14. Для действия выберите **Make a web request**:

15. Для параметров действия нам необходимо вызвать функцию на созданном нами устройстве. Поэтому просто введите параметры, как показано на этом снимке экрана:



16. Вы, конечно, должны изменить URL с ID устройства,который вы установили в коде. После этого создайте рецепт, который теперь должен появиться в вашей учетной записи IFTTT.

17. Теперь вы можете, наконец, протестировать проект, так как IFTTT теперь устанавливает связь между нашими двумя платами. Просто нажмите на кнопку, и вы скоро увидите, что светодиод загорится. Обратите внимание, что это может занять 2-3 секунды, так как информация должна сначала пройти через серверы IFTTT. Вы также можете нажать кнопку еще раз, чтобы выключить светодиод. Вы создали свой первый базовый коммуникационный проект M2M!

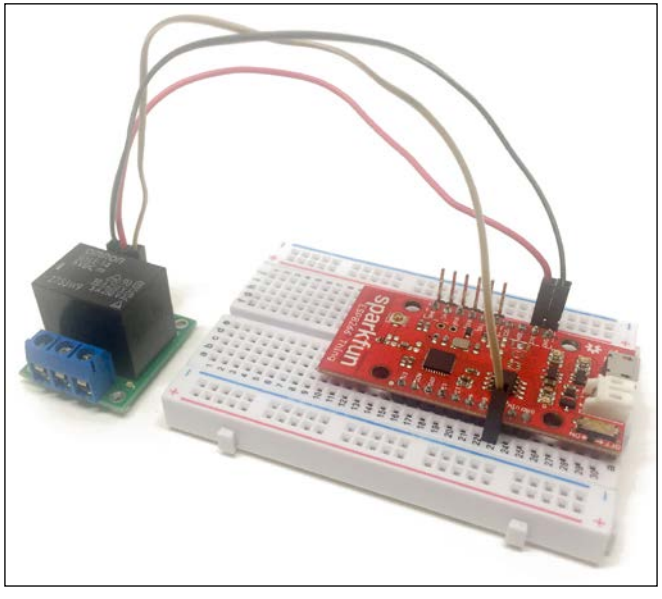
Изготовление светового реле

Теперь, когда мы знаем, как создавать проекты M2M с использованием ESP8266, мы собираемся применить его к простому проекту, в котором вам даже не придется нажимать кнопку. Мы собираемся создать реле с активацией света, которое будет управлять реле, подключенным к одному модулю ESP8266, в зависимости от уровня освещенности, измеренного другим ESP8266. Этот проект будет моделировать часть системы домашней автоматики, которая будет автоматически включать свет, когда на улице темнеет.

Для начала нам нужно собрать обе платы. Начнем с платы, на которой будет размещено реле, так как она самая простая в сборке:

- Просто подключите вывод V_{CC} реле к выводу V_{CC} ESP8266.
- GND к GND
- Вывод SIG или EN реле на вывод 5 ESP8266

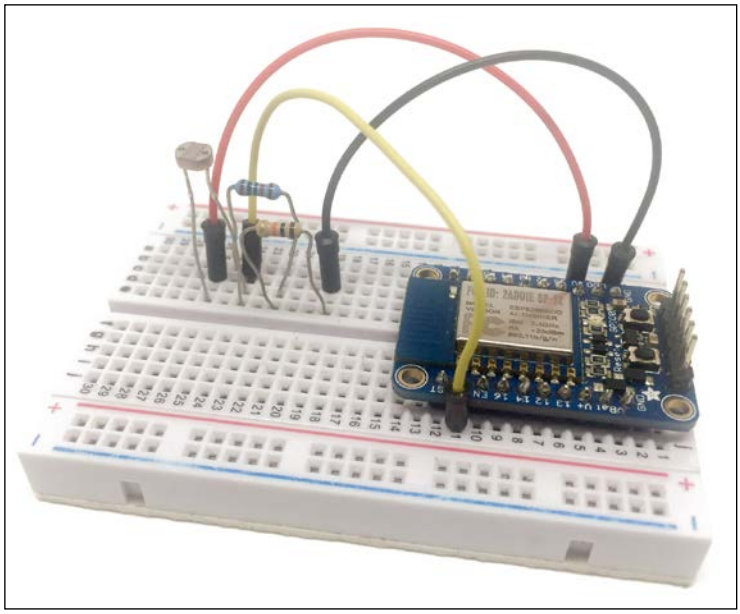
Вот результат для этой платы:



На плату, где будет размещен фотоэлемент, сначала поместите ESP8266 , а затем резистор 10 кОм последовательно с фотоэлементом на макете.

Затем подключите общий вывод фотоэлемента и резистора к выводу A или ADC ESP8266, который является выводом аналогового входа . Затем подключите другой конец фотоэлемента к VCC, а другой конец резистора к GND.

Это окончательный результат для этой платы:



Давайте теперь посмотрим, как настроить эти платы. Для платы реле вы можете просто использовать тот же скетч, что и для платы светодиодов в предыдущем проекте, поскольку мы снова собираемся использовать облачный сервер aREST.

Что касается платы фотоэлементов, она будет очень похожа на плату с кнопками в первом проекте, поэтому здесь мы поговорим только об основных отличиях.

Во-первых, мы устанавливаем переменную, которая будет определять, низкий или высокий уровень освещенности в данный момент.

По умолчанию мы установим его на низкий уровень, присвоив переменной ложное значение:

```
bool lightLevel = false;
```

В функции скетча loop () мы сначала измеряем состояние аналогового вывода и печатаем его на последовательном мониторе:

```
Serial.print("Light level: ");
Serial.println(analogRead(A0));
```

Затем мы проверяем, находится ли уровень освещенности ниже заданного порога, а также было ли предыдущее состояние уровня освещенности высоким. Обратите внимание, что порог здесь произвольный, он просто для иллюстрации связи M2M между двумя платами. Если мы действительно находимся в этих условиях, мы отправляем событие light_level_low:

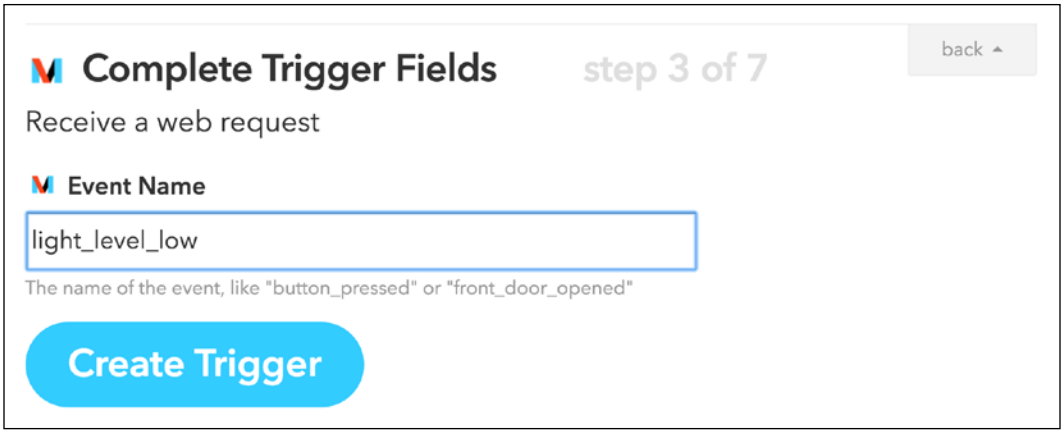
```
if (analogRead(A0) < 700 && lightLevel) {

    lightLevel = false;
    makeRequest("light_level_low");

  }
```

С другой стороны, если уровень освещенности высокий, а раньше мы находились в низком состоянии, мы отправляем событие light_level_high в IFTTT:

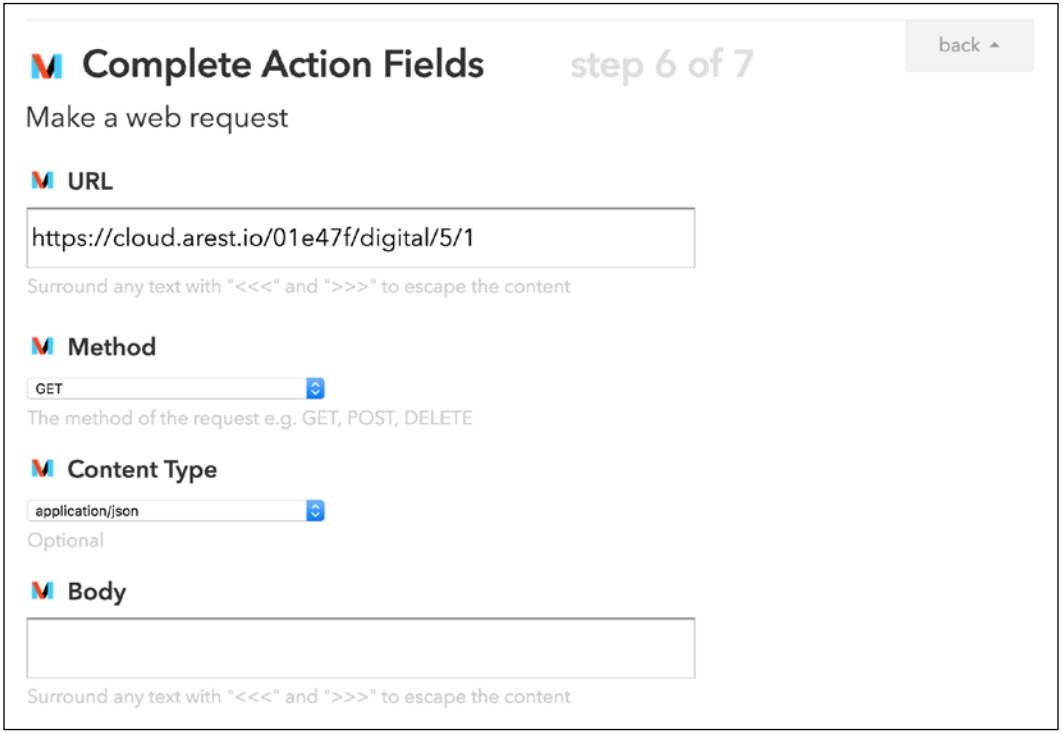
```
if (analogRead(A0) > 800 && !lightLevel) {

    lightLevel = true;
    makeRequest("light_level_high");

  }
```

Теперь вы можете настроить плату с фотоэлементом, используя этот код, который вы также можете взять из папки с кодами. Опять же, теперь нам нужно создать рецепты на IFTTT, чтобы связать две платы.

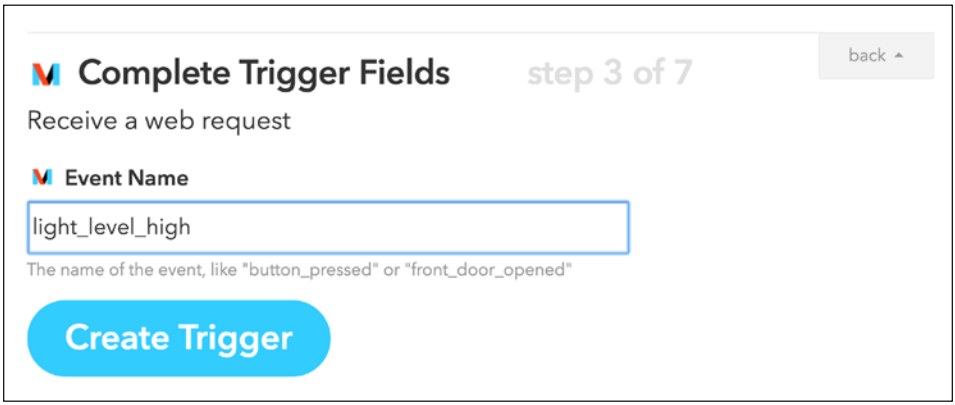
Снова зайдите в IFTTT и создайте новый рецепт. Опять же, используйте канал Maker в качестве запускающего и установите имя события на light_level_low, как определено в скетче:



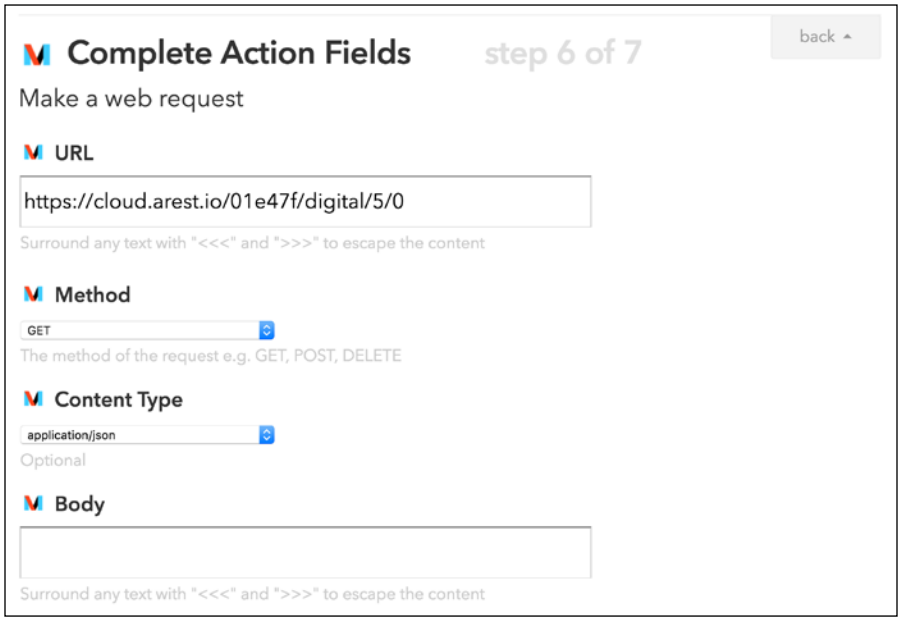
Для действия снова выберите канал Maker и создайте веб-запрос со следующими параметрами:



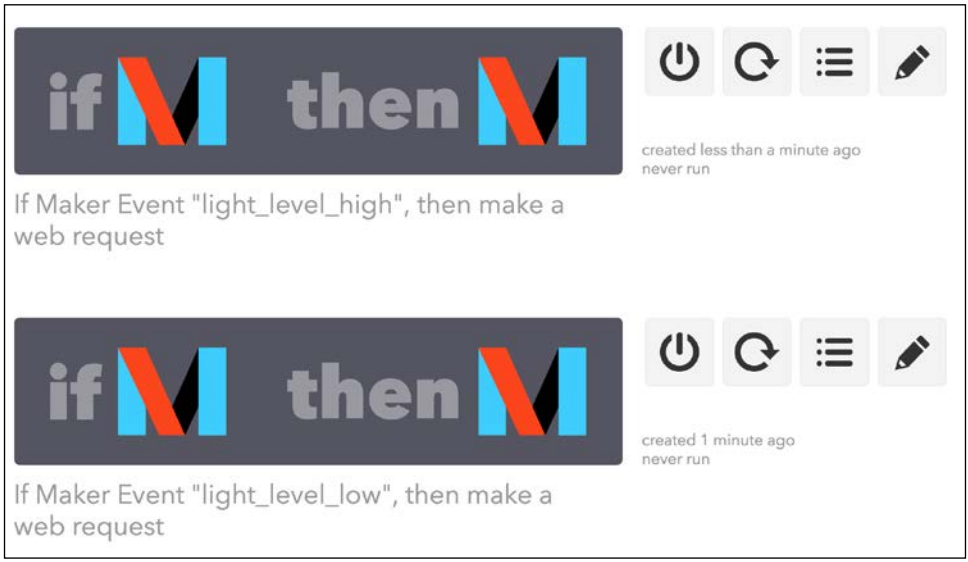
Здесь мы устанавливаем вывод 5 в состояние высокого уровня на целевой плате, так как мы хотим включить реле, если уровень освещенности низкий. Конечно, не забудьте изменить ID вашего устройства в URL-адресе, который будет вызываться IFTTT.
Создайте рецепт и сделайте то же самое с другим сценарием, создав другой рецепт. На этот раз выберите light_level_high в качестве запускающего события:



Для действия введите точно такой же запрос, как и раньше, но на этот раз с 0 в конце, так как мы хотим выключить свет, если уровень освещения снова станет высоким:



Теперь вы должны увидеть оба рецепта активными на панели инструментов IFTTT:



Теперь вы можете протестировать проект! Чтобы имитировать темноту на улице, просто положите руку на фотоэлемент. Вы должны увидеть, что через 2-3 секунды после этого реле должно автоматически переключиться в активное состояние на другой плате. Если убрать руку, реле должно снова выключиться. Обратите внимание: поскольку мы используем M2M связи через облачный сервис, обе платы, конечно, могут быть размещены в разных местах. Например, вы легко можете разместить одну плату снаружи и одну внутри дома, даже если это означает, что они подключены к разным сетям Wi-Fi.

Резюме

Подведем итоги того, чего мы достигли в этом проекте. Мы увидели, как реализовать M2M-связь с помощью чипа ESP8266 Wi-Fi. Мы проиллюстрировали это, построив два простых проекта, в которых IFTTT использовался для связи между платами ESP8266.

Конечно, теперь вы можете использовать все знания для создания ваших собственных проектов Machine-to-Machine. Коммуникации M2M - это очень широкая тема, и с ее помощью можно реализовать очень много вещей. Например, вы можете создать полную систему безопасности на основе облака, состоящую только из модулей ESP8266, взаимодействующих друг с другом, каждый из которых оснащен датчиком движения, что мы также увидим позже в книге.
В следующей главе мы снова будем использовать IFTTT, но для другой цели: отправки автоматических уведомлений из ваших проектов ESP8266.

7

Отправка уведомлений с ESP8266

В этой главе мы увидим, как интегрировать очень важный элемент Интернета вещей с ESP8266: уведомления. Устройства Интернета вещей постоянно предупреждают пользователей, когда происходит что-то важное, или просто через определенные промежутки времени, например, чтобы сообщить данные.

В этой главе мы увидим три сценария. Сначала мы увидим, как отправлять простые уведомления по электронной почте с ESP8266. Затем мы узнаем, как заставить чип регулярно отправлять данные на ваш телефон в виде текстовых сообщений. Наконец, мы увидим, как отправлять оповещения с помощью push-уведомлений. Давайте начнем!

Аппаратные и программные требования

Для этого проекта вам, конечно же, понадобится микросхема ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266.
В качестве датчика я буду использовать здесь датчик DHT11, чтобы проиллюстрировать поведение двух из трех сценариев в этой главе. Однако вы можете использовать любой датчик или даже фиктивные данные для этой главы. Цель действительно научиться отправлять уведомления.

Вам также понадобится USB-модуль FTDI 3,3 В / 5 В для программирования микросхемы ESP8266.

Наконец, вам также понадобятся перемычки и макетная плата.

This is a list of all the components that will be used in this chapter:

- Adafruit ES8266 module (`https://www.adafruit.com/products/2471`)
- FTDI USB module (`https://www.adafruit.com/products/284`)
- DHT11 sensor (`https://www.adafruit.com/products/386`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/9194`)

Что касается программы, если это еще не сделано, вам нужно будет установить последнюю версию ArduinoIDE, которую вы можете получить отсюда:

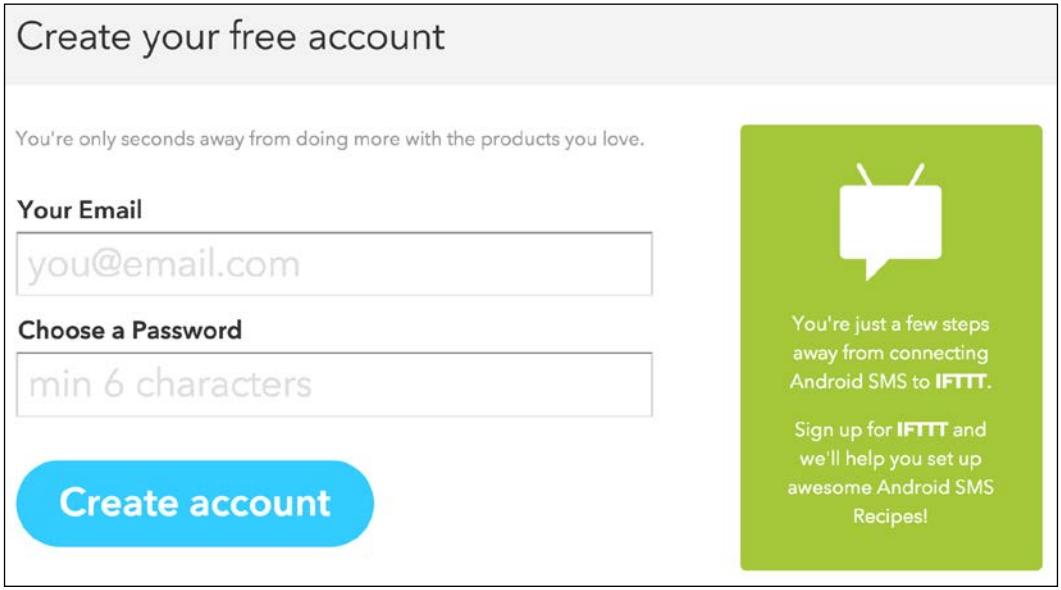
`http://www.arduino.cc/en/Main/Software`

Затем вам понадобится учетная запись IFTTT, которую мы будем использовать для всех проектов в этой главе. IFTTT - это очень крутая веб-служба, которая может связывать две веб-службы с помощью «рецептов», которые активируются триггером (If), который, в свою очередь, запускает действие (the that).

Чтобы создать аккаунт, просто перейдите по ссылке:

`https://ifttt.com/recipes`

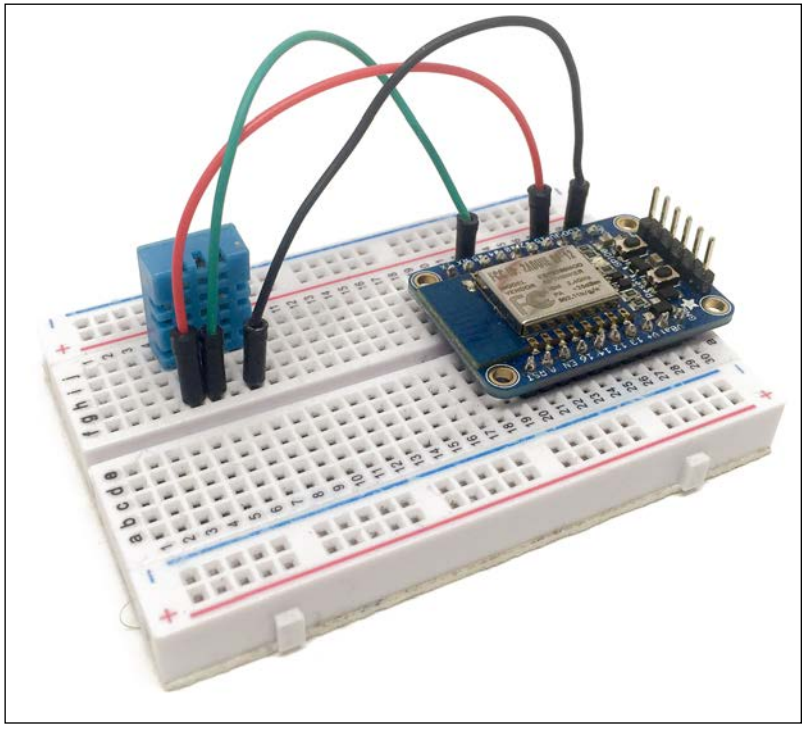
После этого вы сможете создать свою бесплатный аккаунт:



Аппаратная конфигурация

Теперь смонтируем устройство на макетной плате для этого проекта. Поскольку мы уже делали это в предыдущих главах, я отсылаю вас к главе 5 «Взаимодействие с веб-службами», чтобы узнать, как собрать устройство для этого проекта.

Вот как это должно выглядеть в конце:

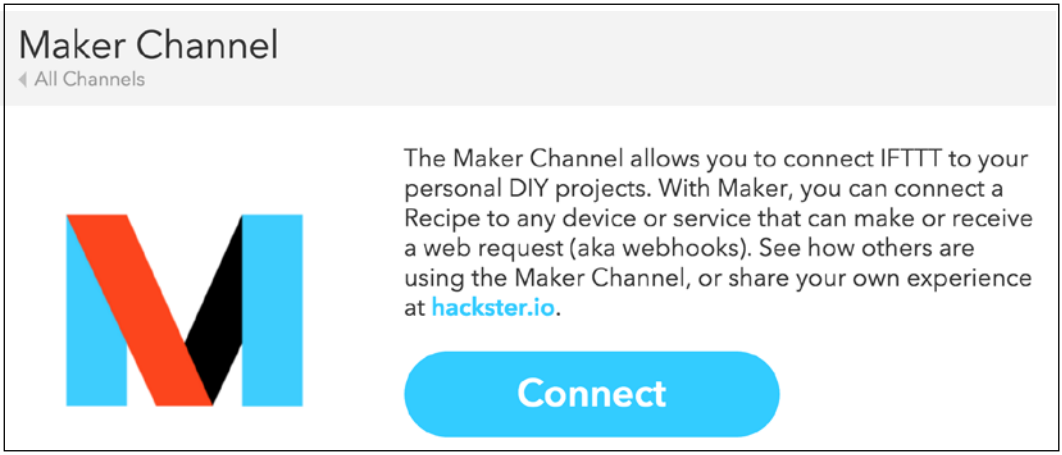


Отправка уведомления по электронной почте

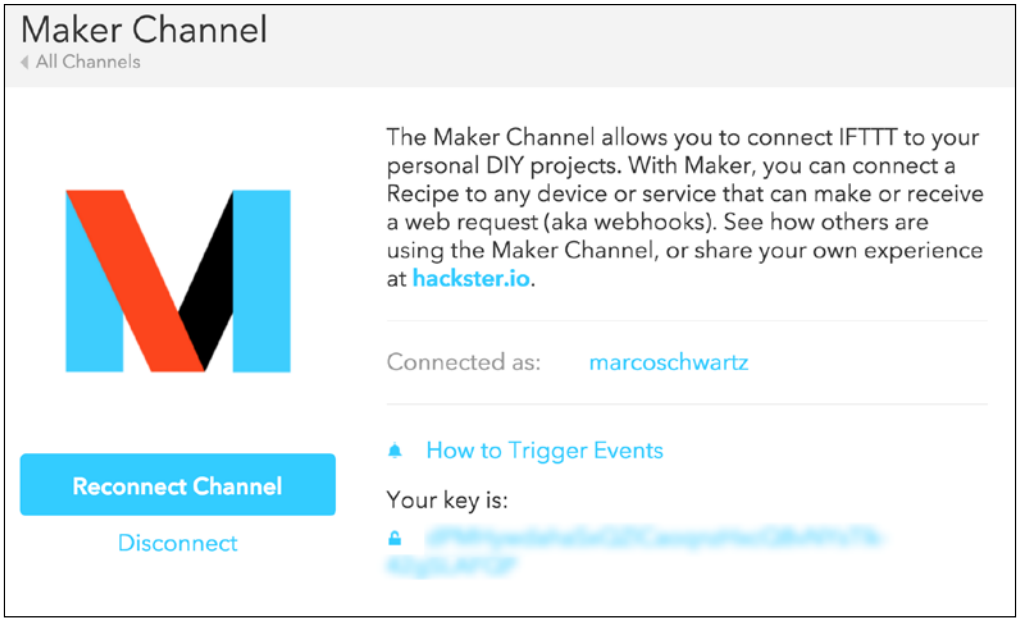
Пришло время сделать наш первый проект: отправка уведомлений по электронной почте! Для этого первого подхода к уведомлениям мы просто будем регулярно отправлять сообщения на выбранный вами адрес электронной почты.

Чтобы подключить ESP8266 к IFTTT, нам понадобится канал Maker, который доступен на IFTTT. Сначала вам нужно перейти на вкладку Channels - Каналы и найти этот канал:

Как только канал откроется на вашем экране, вам необходимо подключить его к своей учетной записи IFTTT:

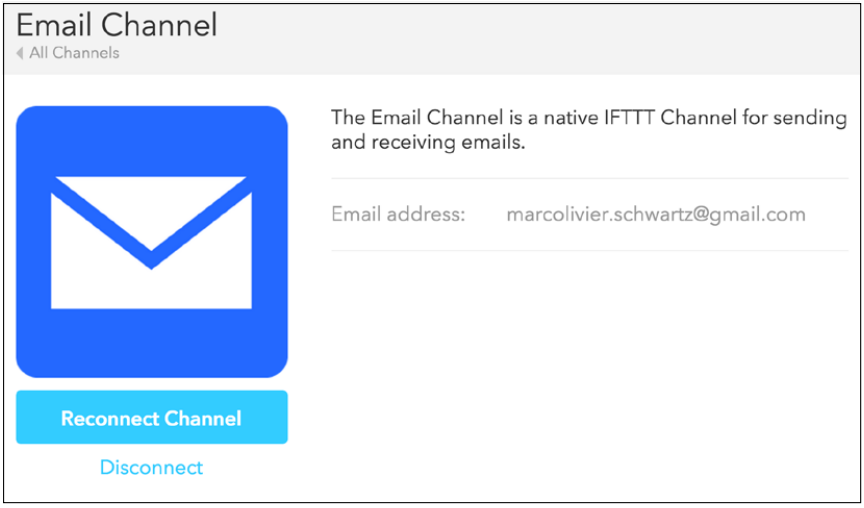


Это позволит вам получить ключ для вашего канала Maker:



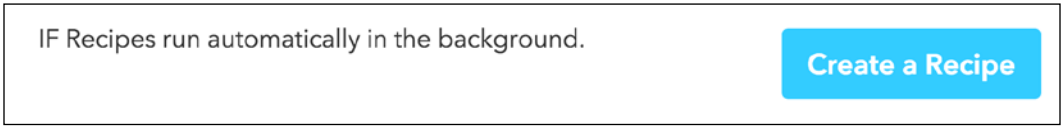
Убедитесь, что этот ключ у вас под рукой; он тебе очень скоро понадобится. Что касается всех каналов, вам нужно сделать это только один раз.

Следующий канал, который мы добавим, - это Email Channel - канал электронной почты:

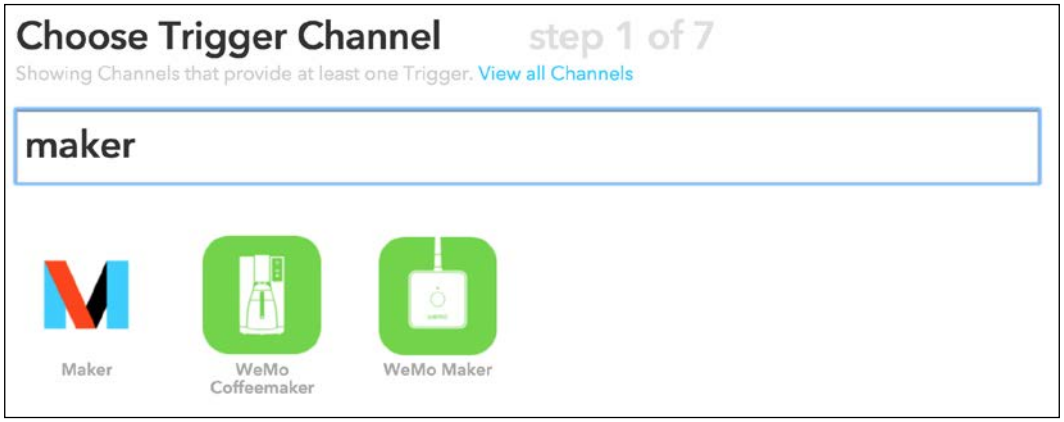


Здесь введите адрес электронной почты, который вы будете использовать для получения тестовых сообщений, приходящих с сервера IFTTT. Обязательно укажите свой e-mail,иначе это будет считаться спамом. Затем мы можем создать рецепт, который свяжет канал Maker с отправкой электронного письма.

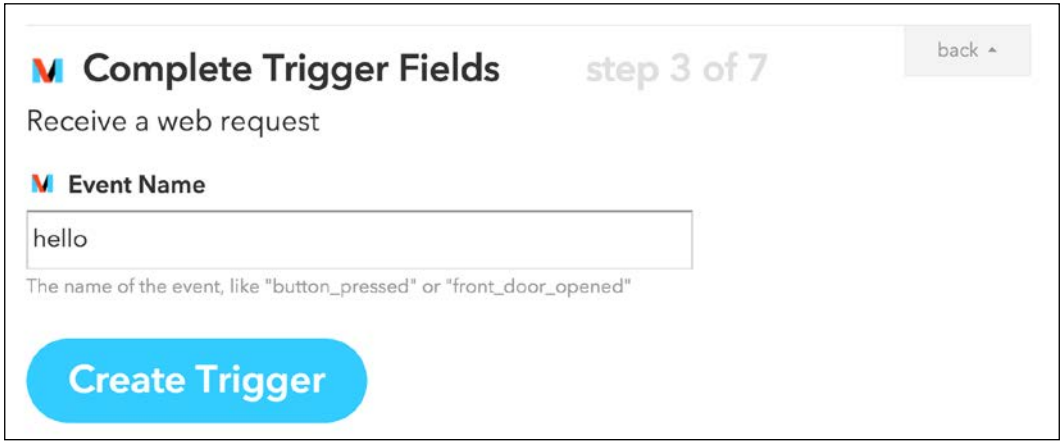
Просто нажмите следующую кнопку на главной странице:



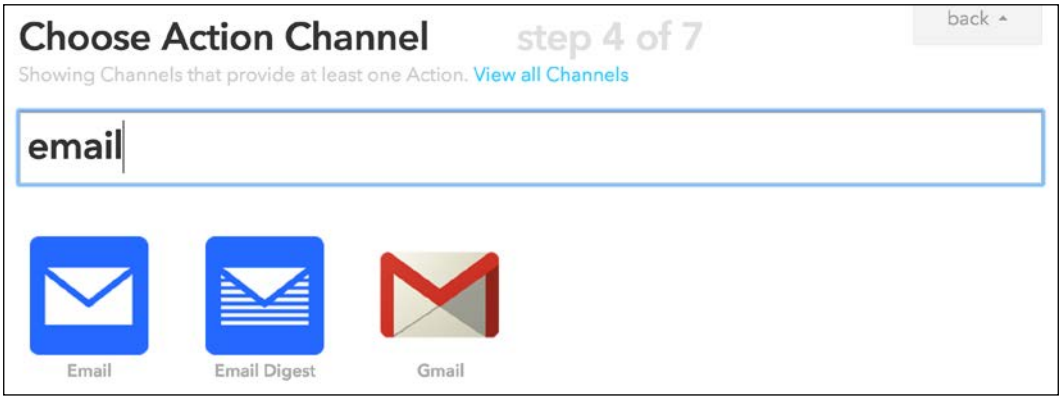
На экране, где вы можете создать новый рецепт, выберите канал Maker, который мы подключили ранее:



В качестве запуска мы просто напишем hello:

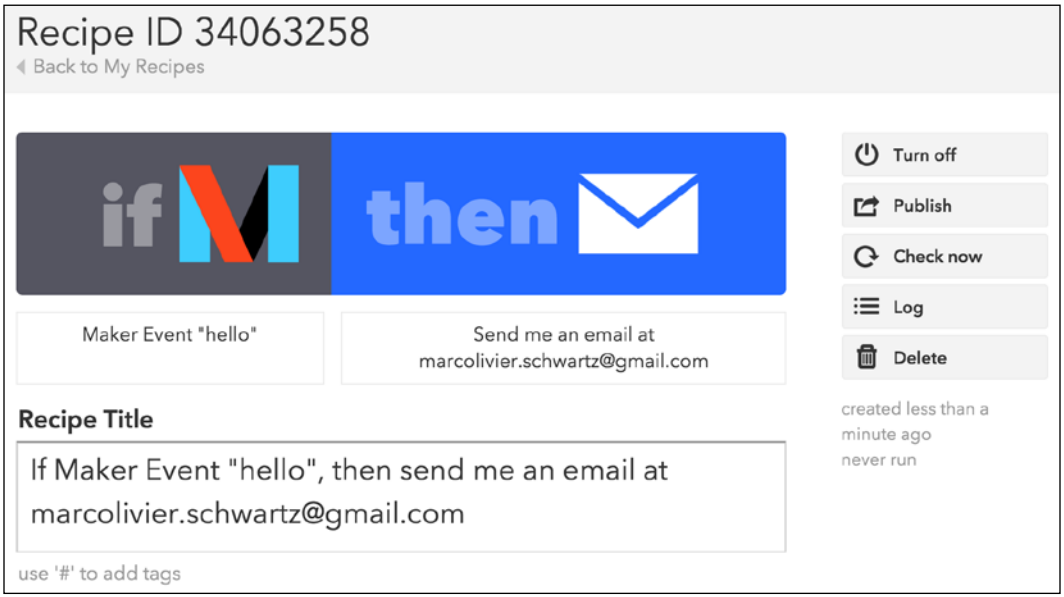


Для Action Channel выберите канал e-mail, который вы добавили ранее:



Затем вас попросят указать в канале действия, какой e-mail вы хотите отправить. Поскольку мы просто хотим протестировать здесь уведомления, просто введите простое сообщение:

Теперь вы можете сохранить рецепт, который появится на панели инструментов IFTTT:



Теперь, когда рецепт активен, создадим скетч Arduino, чтобы фактически активировать этот рецепт. Как обычно, я подробно расскажу о наиболее важных частях скетча:

1. Он начинается с включения библиотеки Wi-Fi ESP8266:

```
#include <ESP8266WiFi.h>
```

2. Затем вам нужно ввести имя Wi-Fi и пароль:

```
const char* ssid     = "wifi-name";
const char* password = "wifi-pass";
```

3. После этого вам нужно ввести некоторую информацию об IFTTT, такую как название события и ключ вашего канала IFTTT Maker:

```
const char* host = "maker.ifttt.com";
const char* eventName   = "hello";
const char* key = "your-key";
```

4. Теперь в функции setup () подключаем ESP8266 к сети Wi-Fi:

```
Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

5. В функции loop () мы сначала подключаемся к серверу IFTTT:

```
WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }
```

6. Затем готовим запрос с названием события:

```
String url = "/trigger/";
  url += eventName;
  url += "/with/key/";
  url += key;
```

7. Затем мы можем отправить запрос на сервер:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  int timeout = millis() + 5000;
  while (client.available() == 0) {
    if (timeout - millis() < 0) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }
```

8. И читаем ответ с сервера:

```
while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
```

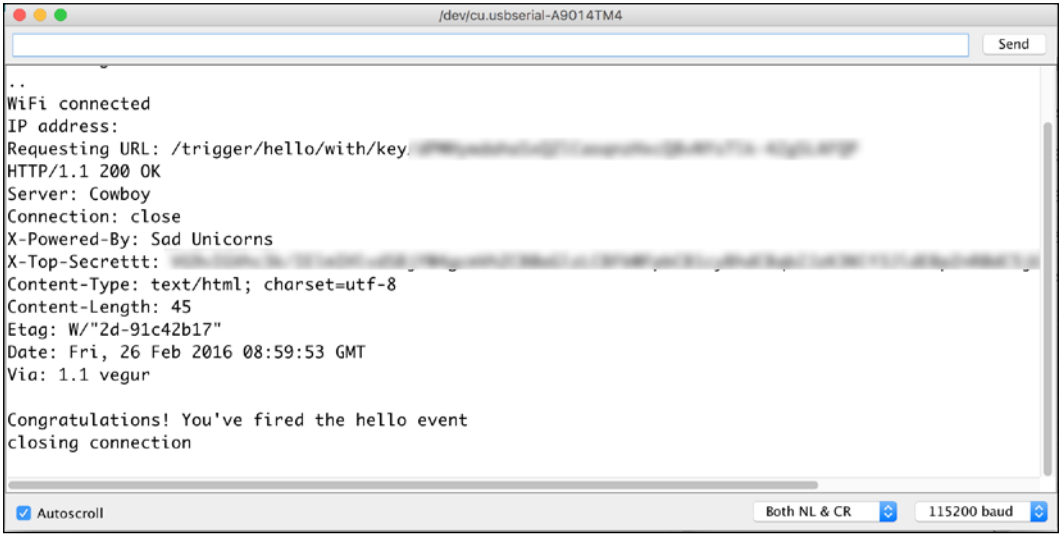
9. Наконец, мы ждем одну минуту перед каждой отправкой:

```
delay(60 * 1000);
```

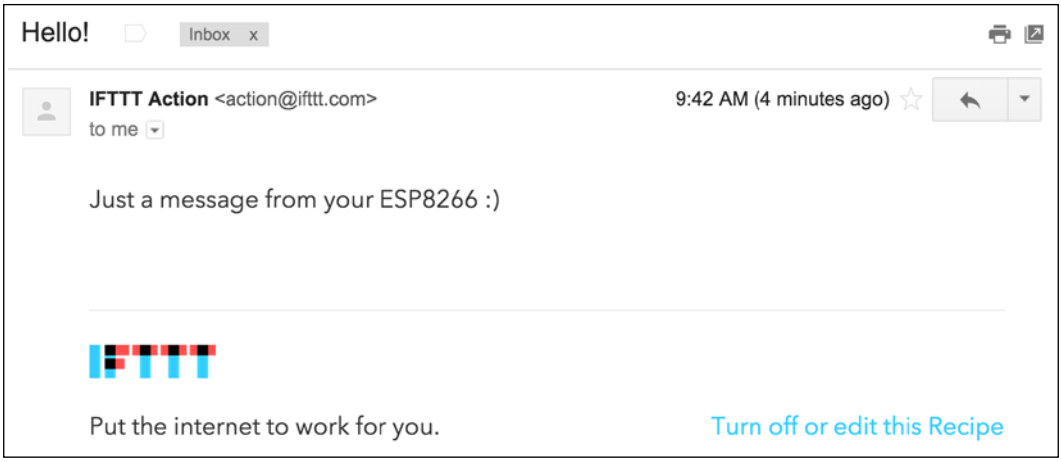
Пришло время, наконец, протестировать проект! Возьмите код из папки с кодами:

`https://github.com/openhomeautomation/iot-esp8266-packt`

Затем измените код, указав свои собственные настройки Wi-Fi и данные IFTTT. Загрузите код на плату. Откройте последовательный монитор, и выдолжны увидеть, что плата подключена к серверам IFTTT и получает подтверждение того, что событие было инициировано:



Теперь проверьте свой почтовый ящик. Вы должны увидеть, что только что получили сообщение от ESP8266:

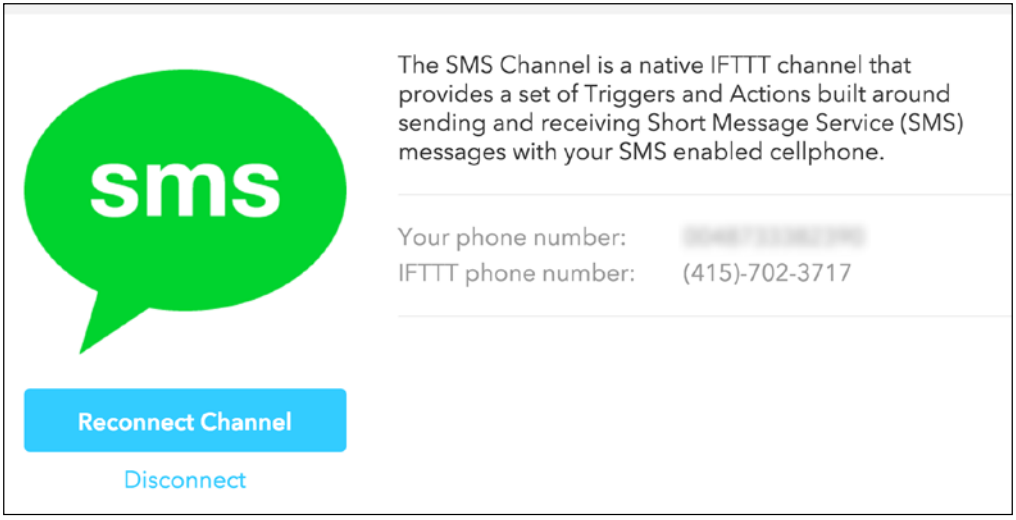


Конечно, не забудьте быстро отключить ESP8266 или выключить рецепт, иначе ваш собственный проект заспамит вас!

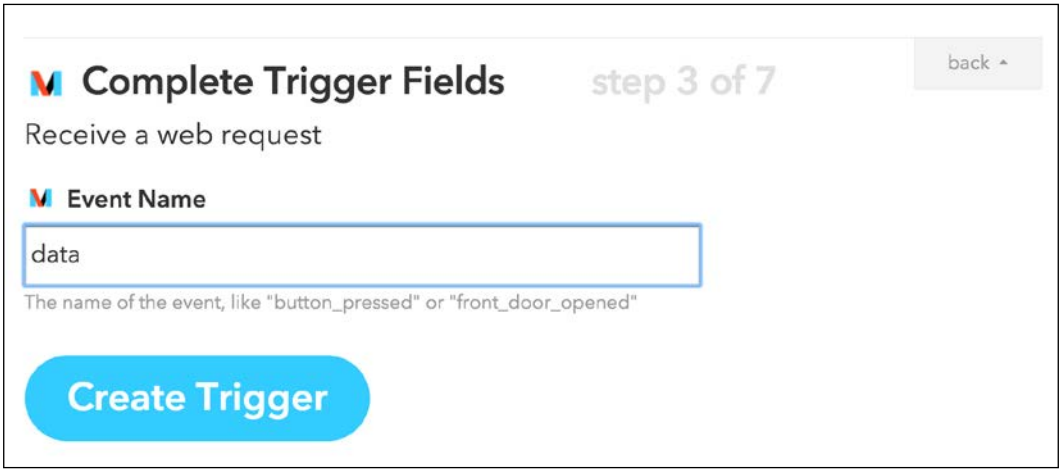
Отправка данных в текстовом сообщении

Теперь мы собираемся использовать то же устройство для создания совершенно другого проекта. Мы снова будем использовать IFTTT для отправки данных измерений с чипа ESP8266 на ваш мобильный телефон:

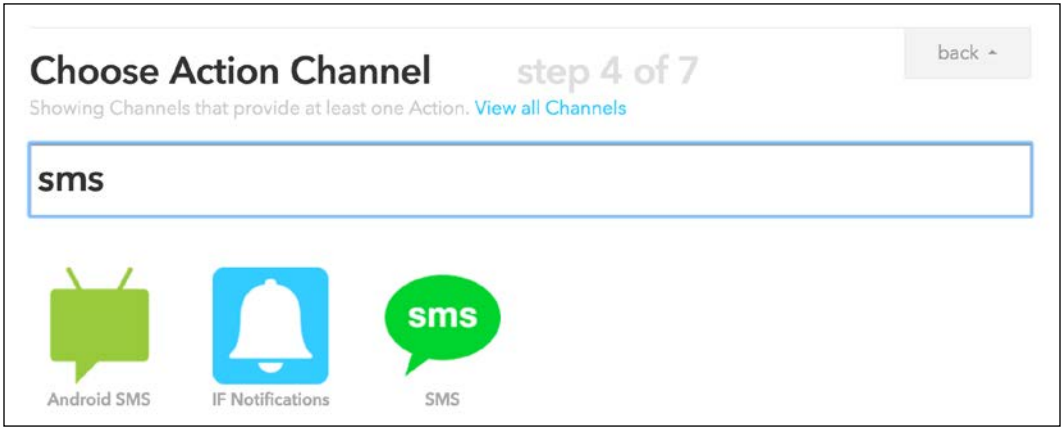
1. Для этого первым делом необходимо подключить канал SMS к вашему аккаунту IFTTT.
   Это очень просто, вам просто нужно будет ввести свой номер телефона:



2. Пришло время создать новый рецепт. На этот раз используйте данные в качестве названия события:



3. Затем выберите вновь созданный канал SMS в качестве действия рецепта:



4. Теперь мы собираемся создать более сложное сообщение, чем раньше, потому чтомы хотим, чтобы в сообщении была информация об измерениях, выполненныхESP8266. Благодаря IFTTT с помощью помощника очень легко добавлять значения, поступающие из канала запуска:

5.Вот как ваше сообщение должно выглядеть в конце:

6. Наконец, создайте рецепт:

Теперь, когда рецепт активен, мы можем перейти к созданию скетча Arduino. Поскольку есть много общих элементов с предыдущим скетчем, здесь я подробно остановлюсь только на важных элементах:

1. Вам необходимо включить библиотеку DHT:

```
#include "DHT.h"
```

2. Затем мы устанавливаем контакт датчика DHT и вводим:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

3. Также создаем экземпляр датчика DHT:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

4. На этот раз мы назовем данные события:

```
const char* eventName   = "data";
```

5. В функции скетча setup () мы инициализируем датчик DHT:

```
dht.begin();
```

6. Затем внутри функции loop () мы измеряем данные с датчика:

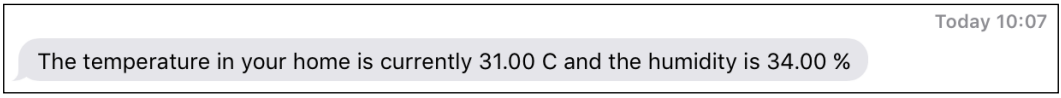
```
float h = dht.readHumidity();
float t = dht.readTemperature();
```

7. После этого мы помещаем в запрос только что измеренные данные:

```
String url = "/trigger/";
  url += eventName;
  url += "/with/key/";
  url += key;
  url += "?value1=";
  url += String(t);
  url += "&value2=";
  url += String(h);
```

Пришло время протестировать скетч! Возьмите его из папки с кодами и обязательно измените скетч с вашими настройками Wi-Fi и данными IFTTT.

Затем загрузите код на плату. Через несколько секунд вы должны получить сообщение на свой мобильный телефон с данными, которые были измерены:



Получение предупреждений через push-уведомления

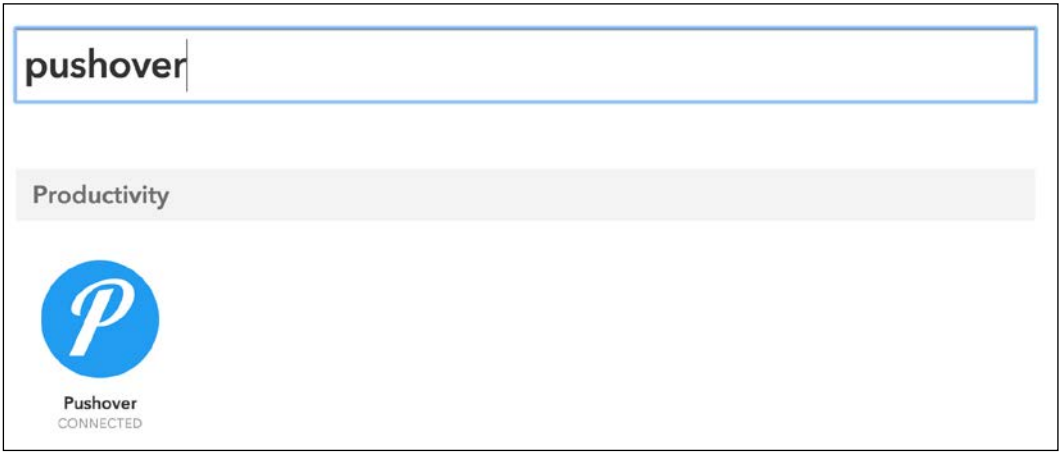
В последнем проекте главы мы увидим, как отправлять третий тип уведомлений: push-уведомления. Они идеально подходят для важных предупреждений, поскольку они будут отображаться прямо на вашем телефоне при срабатывании.

Для этого мы будем использовать приложение под названием Pushover, которое существует для iOS и Android. Сначала вам необходимо создать учетную запись по адресу:

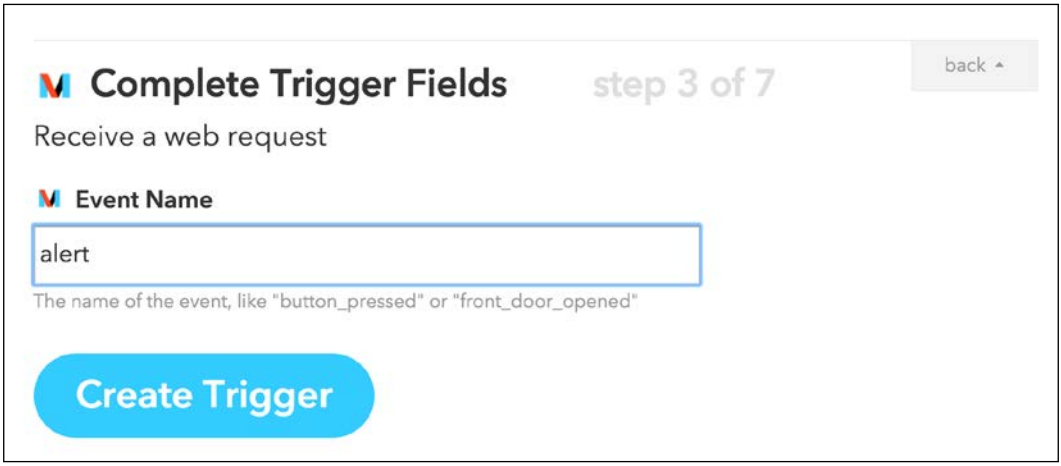
```
https://pushover.net/
```

Внутри панели управления Pushover вам необходимо получить ключ API; вам понадобится это через мгновение, чтобы связать Pushover с вашей учетной записью IFTTT.

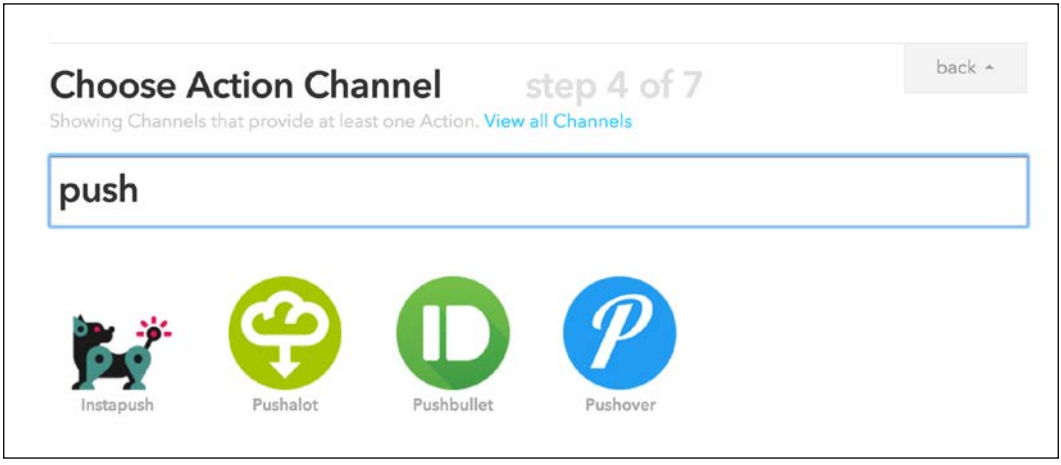
Внутри IFTTT теперь можно подключить канал Pushover:



Теперь мы готовы создать окончательный рецепт этой главы. В качестве названия события на этот раз выберите alert - предупреждение:

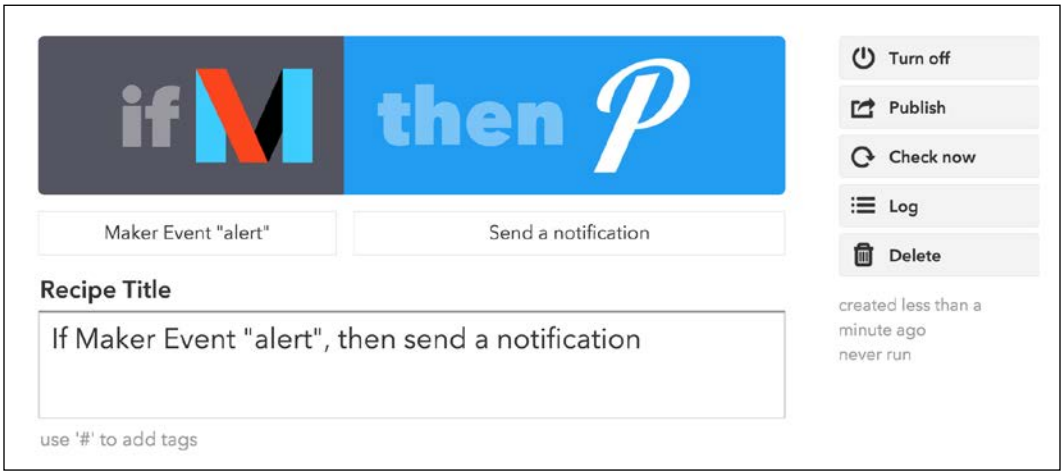


Затем в качестве канала действия выберите канал Pushover:



В качестве сообщения для push-уведомления вы можете выбрать то, что хотите, например сообщение о том, что влажность слишком высока:

Наконец, сохраните рецепт, который теперь должен появиться на вашей панели инструментов:



Теперь настроим плату ESP8266. Поскольку скетч действительно похож на другие скетчи, которые мы уже видели, я расскажу вам только о том, что здесь изменилось.

Первое, что нам нужно изменить, это название события, отправляемого в IFTTT:

```
const char* eventName   = "alert";
```

Затем мы настраиваем плату для отправки предупреждения, когда влажность поднимается выше 30%:

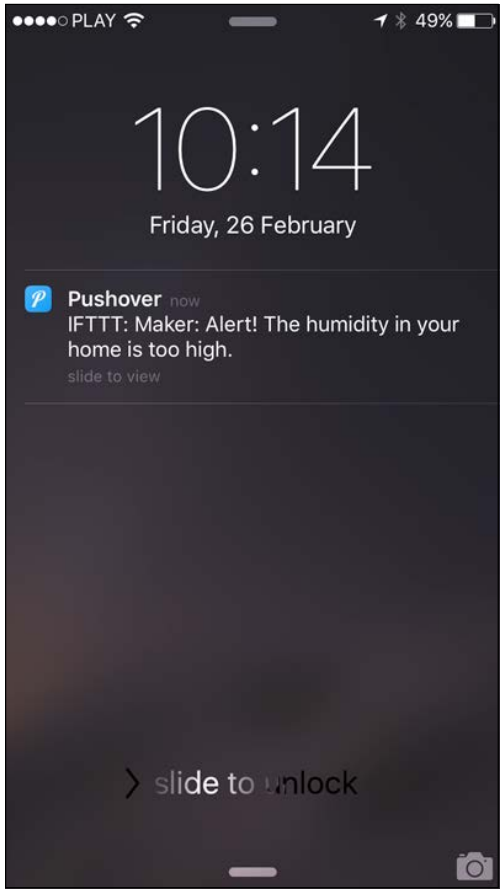
```
if (h > 30.00) {
```

Нам также нужно добавить большую задержку при срабатывании предупреждения, чтобы убедиться, что вы не получаете предупреждениекаждые 5 секунд:

```
delay(10 * 60 * 1000);
```

Наконец-то пора протестировать проект! Возьмите код из папки с кодами и измените настройки Wi-Fi, а также данные IFTTT. Затем загрузите код на плату.

Если влажность действительно выше порогового значения, вы вскоре получите уведомление на свой телефон:



Это намного лучше, чем просто электронное письмо, поскольку вы увидите его, даже если в данный момент не проверяете электронную почту или текстовые сообщения. Таким образом, он идеально подходит для срочных предупреждений, и теперь вы знаете, как отправлять эти предупреждения прямосо своего ESP8266.

Резюме

В этой главе мы увидели, как отправлять автоматические уведомления с вашего ESP8266 на вашу электронную почту, ваш телефон с помощью текстовых сообщений и push-уведомлений. Это позволяет нам либо отправлять данные обратно на наши мобильные устройства, либо создавать оповещения на основе данных измерений.

Вы, конечно, можете адаптировать то, что вы узнали в этом проекте, к своим потребностям и создавать оповещения, поступающие от вашего ESP8266 (или от нескольких модулей!), Которые будут немедленно сообщать вам о любых изменениях в том, что вы хотите отслеживать.

В следующей главе мы собираемся взять все, что мы узнали до сих пор, и создать новый проект с использованием ESP8266: дверной замок, которым вы можете управлять из облака.

8

Управление дверным замком из облака

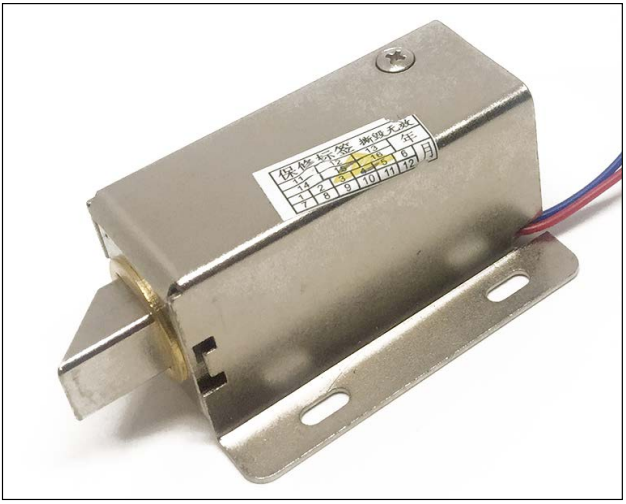
Это будет первый проект в этой книге, в котором мы возьмем все, чему мы научились до сих пор, и интегрируем это, чтобы создать законченный проект Интернета вещей, который вы действительно сможете использовать в своей жизни.
Для этого проекта мы собираемся использовать дверной замок, которым мы сможем управлять из любой точки планеты с помощью чипа ESP8266 Wi-Fi. Мы также увидим, как интегрировать уведомления прямо в программное обеспечение, которое вы создадите, чтобы вы получали push-уведомление в случае, если кто-то попытается открыть дверной замок без вашего ведома.

Аппаратные и программные требования

Для этого проекта вам, конечно же, понадобится микросхема ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266.

Для замка я использовал дверной замок на базе соленоида 12 В постоянного тока. Вы, конечно, можете использовать любую другую эквивалентную блокировку, представленную на рынке. Для использования замка вам понадобится несколько компонентов: NPN-транзистор, защитный диод и резистор 1 кОм. Вам также понадобится блок питания 12 В постоянного тока, который можно подключить к макетной плате.

Это замок, который я буду использовать в этом проекте:



Вам также понадобится USB-модуль FTDI 3,3 В / 5 В для программирования микросхемы ESP8266.

Наконец, вам также понадобятся перемычки и макетная плата

Это список всех компонентов, которые будут использоваться в этой главе:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- Соленоид LockStyle
- NPN транзистор
- Резистор
- Диод
- Макетная плата
- Перемычки

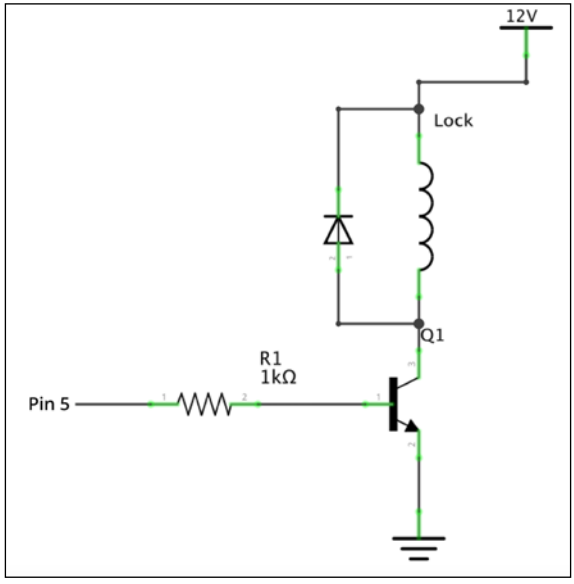
Что касается программы, если это еще не сделано, вам нужно будет установить последнюю версию Arduino IDE, которую вы можете получить отсюда:

`http://www.arduino.cc/en/Main/Software`

Then, for the last part of the chapter, you will also need an IFTTT account. If you don't have one, please refer to the *Chapter 5*, *Interacting With Web Services* to learn how to create your IFTTT account.

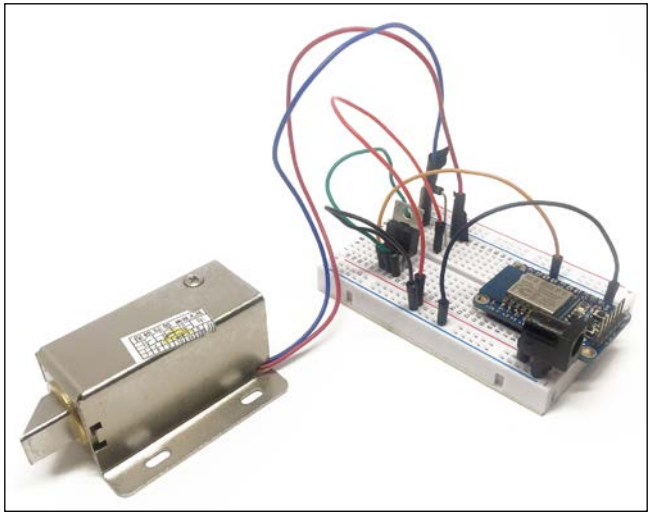
Аппаратная конфигурация

Конфигурация для этого проекта довольно сложна, поэтому я включил схему, чтобы помочь вам:



По этой схеме разместите компоненты на макетной плате

Вот как это должно выглядеть в конце:



Настройка платы ESP8266

Пришло время настроить чип Wi-Fi ESP8266, чтобы он мог принимать команды, поступающие из облака. Это то, что мы уже видели ранее, но всегда полезно помнить основы, так как позже в этой главе мы напишем более сложный скетч:

1. Сначала нам нужно включить необходимые библиотеки:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

2. Затем мы объявляем клиента Wi-Fi и клиента PubSub (MQTT):

```
WiFiClient espClient;
PubSubClient client(espClient);
```

3. После этого создаем экземпляр библиотеки aREST:

```
aREST rest = aREST(client);
```

4. Также вам необходимо ввести в скетч свое имя Wi-Fi и пароль:

```
const char* ssid = "wifi-ssid";
const char* password = "wifi-pass";
```

5. Вы также можете дать своему устройству имя:

```
rest.set_id(device_id);
rest.set_name("door_lock");
```

6. Наконец, в функции скетча loop () мы обрабатываем соединение, поступающее из облака:

```
rest.handle(client);
```

Пришло время настроить плату! Просто возьмите код из папки с кодами, обязательно измените настройки Wi-Fi и пароль, загрузите код на плату и оставьте его там; мы будем использовать приборную панель для фактического управления замком.

Управление замком из облака

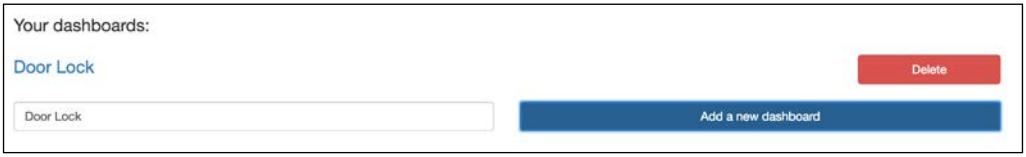
Пришло время посмотреть, как управлять дверным замком из облака. Опять же, мы воспользуемся функцией панели инструментов aREST, чтобы быстро создать облачную панель мониторинга для нашего проекта:

1. Просто перейдите поссылке:

```
http://dashboard.arest.io/
```

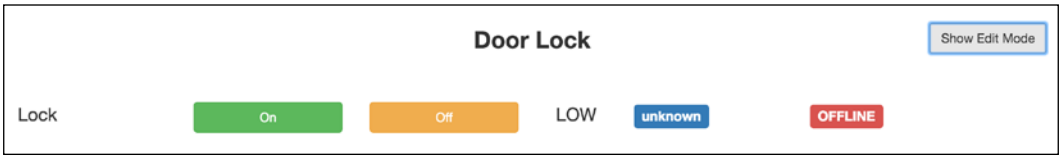
2. После создания аккаунта вы сможете создать свою первую панель управления:

3. После того, как он будет создан, нажмите на имя новой панели инструментов:



4. Оттуда вы сможете создать новый элемент. Поскольку нам просто нужны кнопки on/off, настройте вашу панель со следующими параметрами: назовите элемент Lock, введите идентификатор устройства проекта и выберите соответствующий цифровой PIN-код (5). Все это показано на этом скриншоте:

5. Конечно, не забудьте заменить идентификатор устройства на идентификатор вашего собственного устройства. Как только это будет сделано, вы должны увидеть новый элемент управления на панели инструментов:



Теперь вы можете, наконец, попробовать проект; просто нажмите кнопку «On» и замок сразу же активируется. Теперь вы можете заблокировать или открыть дверь из любой точки мира! Просто убедитесь, что вы не нажимаете разные кнопки слишком быстро. В отличие, например, от светодиода, дверной замок срабатывает через некоторое время (одна или две секунды).

Получение уведомлений при открытии замка

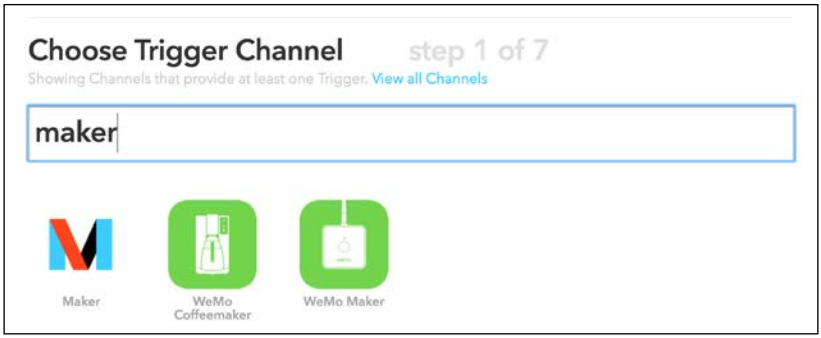
Управлять блокировкой из облака - это здорово, но узнать ее текущее состояние можно, только открыв панель управления на своем компьютере или мобильном телефоне. Но что, если вы находитесь в дороге, а замок установлен в довольно важной двери в вашем доме?

Вы хотите, чтобы вас предупредили, когда дверь откроется.

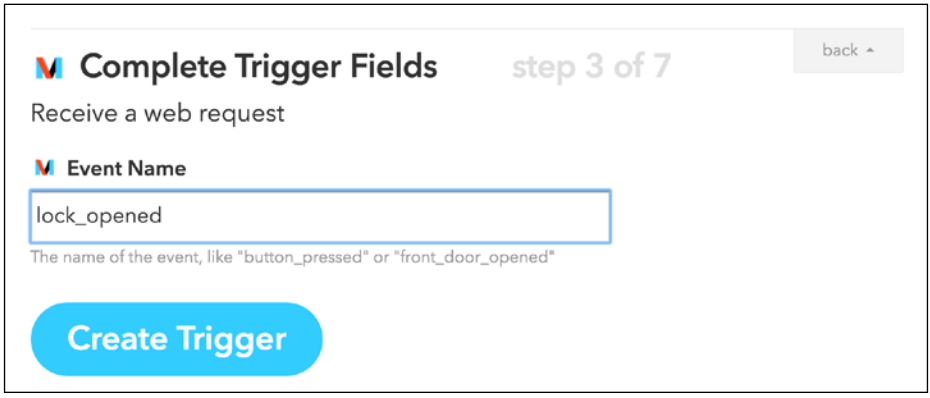
Это именно то, что мы собираемся сделать сейчас, используя IFTTT. Мы собираемся настроить плату так, чтобы она отправляла уведомления на ваш смартфон при открытии дверного замка:

1. Сначала перейдите в IFTTT и добавьте два канала, если это еще не было сделано: канал Maker и канал Pushover. Также установите приложение Pushover на свой смартфон. Чтобы узнать больше об этом, обратитесь к Главе 7, Отправка уведомлений с ESP8266.

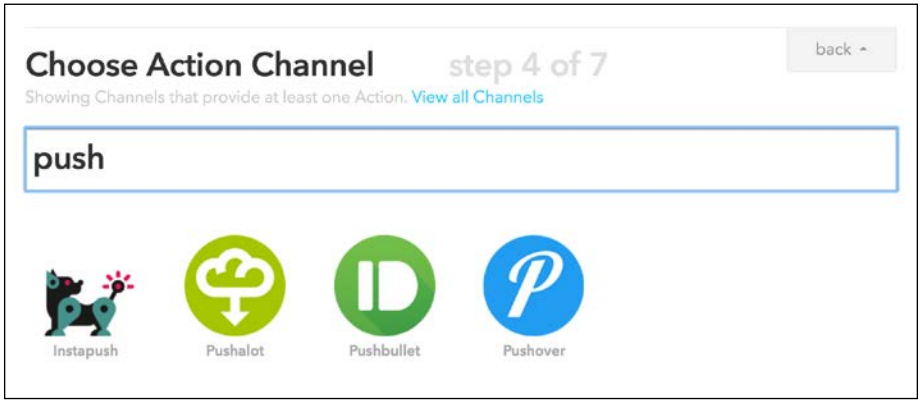
2. Затем создайте новый рецепт и выберите канал Maker в качестве запуска:



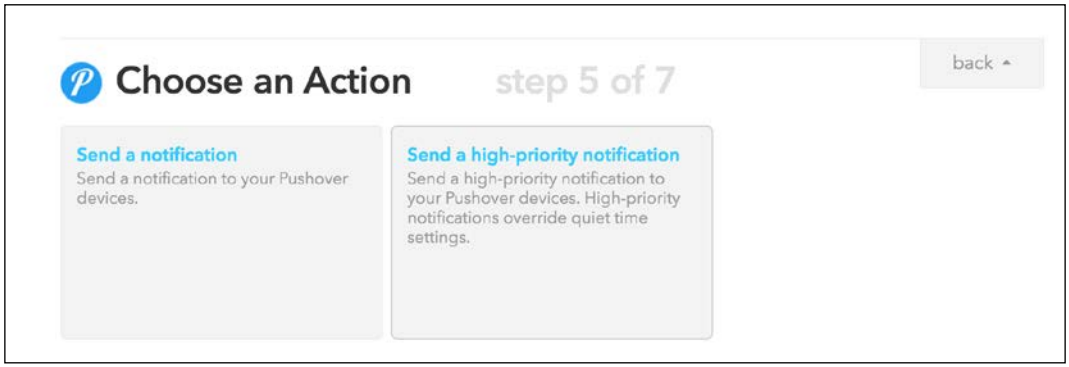
3. Здесь нам нужно использовать канал Maker, так как он позволит нам использовать собственные проекты, такие как наш, с IFTTT. В качестве запуска для канала поместите событие lock_opened:



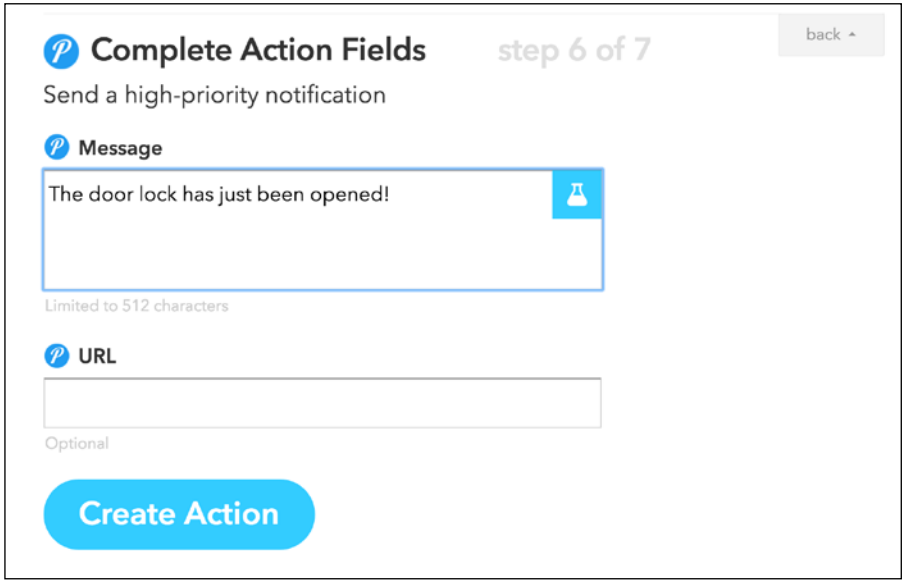
4. В качестве действия просто найдите канал Pushover, который мы снова будем использовать для получения уведомлений от IFTTT:



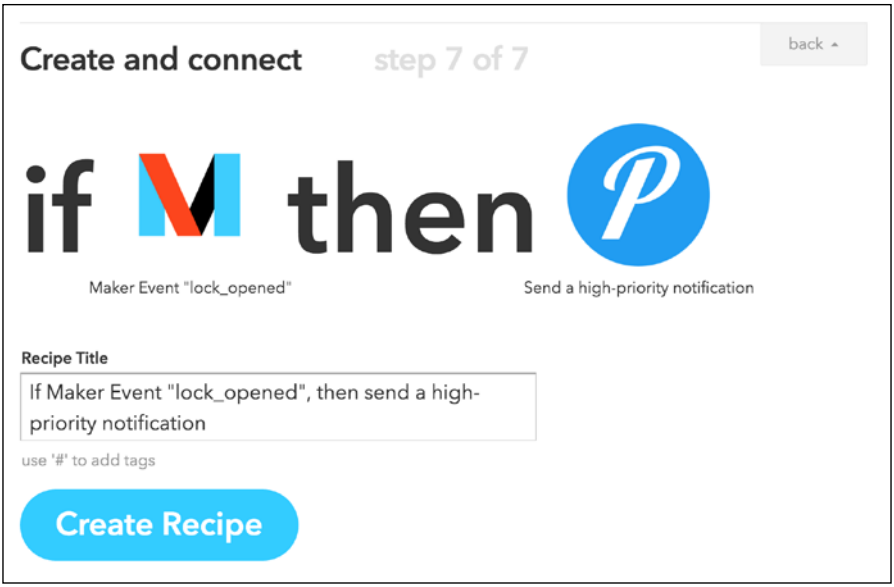
5. Затем выберите желаемый тип уведомления:



6. Это просто укажет IFTTT,должен ли он отправлять обычные -Send a notification или высокоприоритетные уведомления - Send a high-priority notification в ваше приложение Pushover. После этого поместите сообщение,которое вы хотите получать каждый раз при наступлении события:



7. Наконец, подтвердите создание канала:



8. Давайте посмотрим, что нам нужно добавить в предыдущий код для интеграции уведомлений. Во-первых, нам нужно определить две переменные для хранения текущего состояния блокировки:

```
bool lockStatus;
bool previousLockStatus;
```

9. Затем внутри функции скетча setup () мы считываем данные спина 5, чтобы проверить, активирована ли блокировка изначальноили нет:

```
lockStatus = digitalRead(5);
previousLockStatus = lockStatus;
```

10. После этого внутри функции loop () мы снова считываем состояние с вывода 5:

```
lockStatus = digitalRead(5);
```

11. Если он изменился по сравнению с последним чтением, и если блокировка открыта (что означает, что на выводе находится НИЗКИЙ статус), мы отправляем уведомление через IFTTT:

```
if (lockStatus != previousLockStatus && lockStatus == 0) {

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

```
    // Use WiFiClient class to create TCP connections
    WiFiClient client;
    const int httpPort = 80;
    if (!client.connect(host, httpPort)) {
      Serial.println("connection failed");
      return;
    }

    // We now create a URI for the request
    String url = "/trigger/";
    url += eventName;
    url += "/with/key/";
    url += key;

    Serial.print("Requesting URL: ");
    Serial.println(url);

    // This will send the request to the server
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    int timeout = millis() + 5000;
    while (client.available() == 0) {
      if (timeout - millis() < 0) {
        Serial.println(">>> Client Timeout !");
        client.stop();
        return;
      }
    }

    // Read all the lines of the reply from server and print them
to Serial
    while(client.available()){
      String line = client.readStringUntil('\r');
      Serial.print(line);
    }

    Serial.println();
    Serial.println("closing connection");

    // Set previous status
    previousLockStatus = lockStatus;

  }
```

Чтобы узнать больше о том, как отправлять уведомления через IFTTT, ознакомьтесь с Главой 7, Отправка уведомлений с ESP8266.

12. Пришло время протестировать проект! Загрузите код на плату, а затем вернитесь к онлайн-панели инструментов, которую вы использовали ранее для управления замком. На этот раз каждый раз, когда вы открываете замок,вы должны сразу получать уведомление на свой смартфон.

Таким образом, даже если вы поделились своей приборной панелью с кем-то другим, вы получите уведомление, когда они откроют дверной замок.

Резюме

В этой главе мы увидели, как использовать все, что мы узнали до сих пор в этой книге, для создания проекта, который вы можете использовать в своем доме: дверной замок с облачным управлением. Мы увидели, как управлять замком из любой точки мира, а также как заставить проект автоматически отправлять уведомления на ваш смартфон при открытии замка.

Есть много способов улучшить этот проект, используя то, что вы узнали из этой главы.

Один из способов - это, конечно, добавить несколько блокировок на одну и ту же панель инструментов и управлять ими из любого места. Вы также можете добавить в проект функцию таймера и автоматически открывать дверь между заданными интервалами времени (например, для входа уборщицы).

В следующей главе мы увидим, как использовать свои знания о микросхеме Wi-Fi ESP8266 для создания еще одного классного проекта IoT: физического тикера Биткойн.

9

Создание физического биткойн-тикера

В этой книге мы уже видели, что микросхему Wi-Fi ESP8266 можно использовать во многих различных ситуациях в Интернете вещей. В этой главе мы собираемся использовать его для довольно экзотического проекта IoT: получение цены биткойна в реальном времени и отображение этой цены на небольшом OLED-экране.

Сначала мы увидим, что такое Биткойн, и как мы можем узнать цену Биткойна через веб-сервис. Затем мы настроим наш биткойн-билет и отобразим текущую цену биткойнов на OLED-дисплее. Наконец, мы добавим в проект несколько светодиодов, чтобы визуально проверить, идет ли цена вверх или вниз. Давайте начнем!

Что такое биткойн?

Прежде чем мы фактически начнем строить сам проект, давайте сначала поговорим о Биткойне, если вы еще не знаете, что это такое.

Биткойн - это виртуальная валюта, появившаяся еще в 2009 году. В отличие от других виртуальных валют, существовавших до того момента, Биткойн защищен глобальной сетью компьютеров, работающих с одним и тем же распределенным реестром, называемым Блокчейн, который гарантирует каждую совершенную транзакцию.

С тех пор многие люди и компании по всему миру начали принимать биткойны в качестве способа оплаты. Биткойн также принимается в обычных магазинах; например, в магазинах, которые отображают знак Bitcoin Accepted Here:

Конечно, биткойн также продается на биржах и, следовательно, имеет цену в долларах США и в большинстве основных валют. Например, на момент написания этой книги один биткойн стоил 417 долларов США.

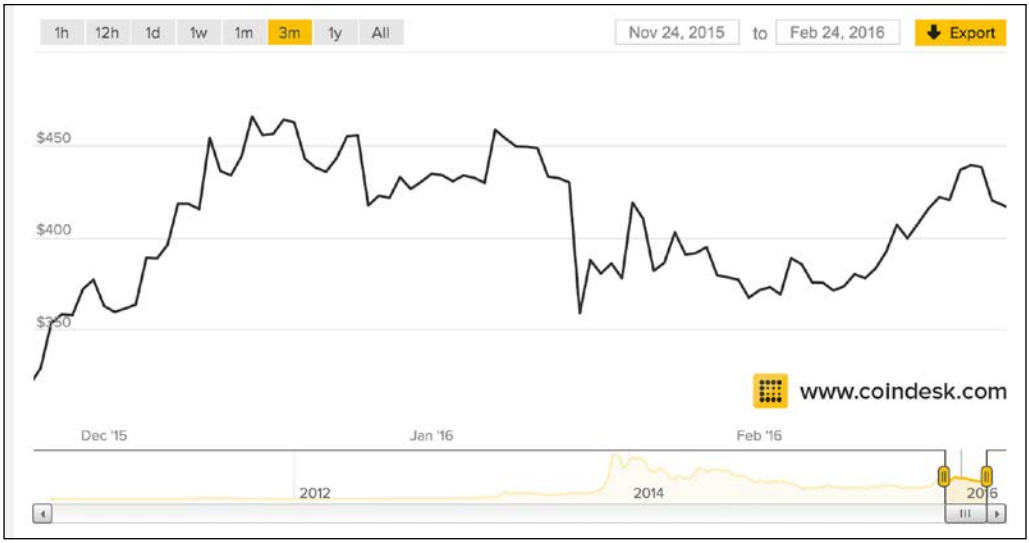
Поэтому было бы довольно интересно создать проект, чтобы узнать, сколько стоит биткойн в настоящее время , с использованием чипа ESP8266 Wi-Fi. Именно этим мы и займемся в этой главе.

Биткойн-сервисы онлайн

Прежде чем начать работу с онлайн-сервисами Биткойн, нам нужно знать, как на самом деле узнать текущую цену (в долларах США) Биткойн.
В Интернете довольно легко узнать цену или историческую стоимость биткойнов.
Например, соответствующий веб-сайт - Coindesk:

```
http://www.coindesk.com/price/
```

На этом веб-сайте у вас есть доступ к текущей цене биткойнов, а также к красивым графикам, показывающим историческую ценность биткойнов:

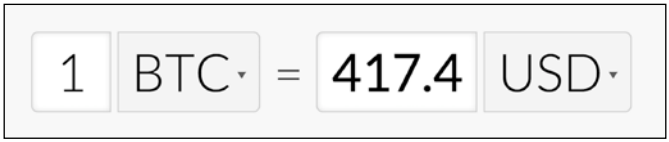


Это хорошо, но доступно только через веб-сайт, и в любом случае это слишком сложно для нашей маленькой платы ESP8266.

Лучшее решение - просто иметь так называемый тикер биткойнов, который просто отображает стоимость биткойнов в реальном времени. Хороший пример такого тикера можно найти по следующему URL-адресу:

```
http://preev.com/btc/usd
```

Вы увидите текущую стоимость биткойнов, которую вы можете отображать в нескольких валютах:



Это то, что мы хотим создать для нашего проекта: простой тикер биткойнов, который отображает текущую стоимость биткойнов в долларах США.

Однако мы не можем получить доступ к этому напрямую с ESP8266. Что нам нужно, так это API (интерфейс прикладного программирования), который будет возвращать текущую цену биткойнов способом, который может быть обработан нашим чипом.

Лучший API, который я знаю, чтобы получить эту цену, также от Coindesk, и его можно получить по адресу:

```
http://api.coindesk.com/v1/bpi/currentprice.json
```

Вы действительно можете попробовать это в веб-браузере. На это вы получите аналогичный ответ:

```
{
   "time":{
      "updated":"Feb 24, 2016 08:15:00 UTC",
      "updatedISO":"2016-02-24T08:15:00+00:00",
      "updateduk":"Feb 24, 2016 at 08:15 GMT"
   },
   "disclaimer":"This data was produced from the CoinDeskBitcoin Price
Index (USD). Non-USD currency data converted using hourly conversion
rate from openexchangerates.org",
   "bpi":{
      "USD":{
         "code":"USD",
         "symbol":"&#36;",
         "rate":"416.4970",
         "description":"United States Dollar",
         "rate_float":416.497
      },
      "GBP":{
         "code":"GBP",
         "symbol":"&pound;",
         "rate":"298.0036",
         "description":"British Pound Sterling",
         "rate_float":298.0036
      },
      "EUR":{
         "code":"EUR",
         "symbol":"&euro;",
         "rate":"378.1955",
         "description":"Euro",
         "rate_float":378.1955
      }
   }
}
```

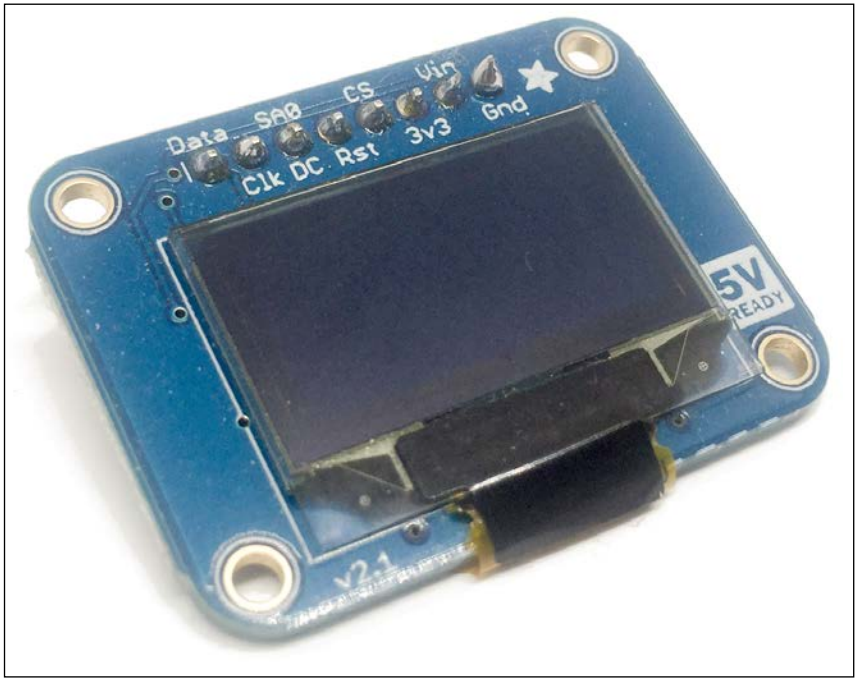
Вы можете видеть, что этот API возвращает текущую цену биткойнов в нескольких валютах.

Это то, что мы будем использовать в нашем проекте, чтобы узнать цену биткойнов.

Аппаратные и программные требования

Давайте теперь посмотрим, какие компоненты необходимы для этого проекта. Вам, конечно, понадобится чип ESP8266. Что касается большей части этой книги, я использовал модуль Adafruit Huzzah ESP8266, но здесь подойдет любой модуль ESP8266 .

Вам также понадобится OLED-дисплей для проекта. Я использовал монохромный OLED-дисплей 128x64 пикселей, используя драйвер SSD1306, который, как мне известно, является единственным совместимым с ESP8266:



В последней части проекта я использовал два светодиода, красный и зеленый, а также резисторы на 330 Ом.

Вам также понадобится USB-модуль FTDI 3,3 В / 5 В для программирования микросхемыESP8266.

Это список всех компонентов, которые будут использоваться в этом проекте:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- LEDx 2
- Резистор 330 Ом x 2
- OLED-дисплей 128x64 пикселей с драйвером SSD1306

- Макетная плата
- Перемычки

Что касается программы, если это еще не сделано, вам необходимо установить последнюю версию Arduino IDE.

Затем вам понадобится библиотека для управления OLED-экраном SSD1306. Я рекомендую загрузить ее из диспетчера библиотек Arduino, но вы также можете получить его из следующего репозитория:

```
https://github.com/squix78/esp8266-oled-ssd1306
```

Конфигурация аппаратной части

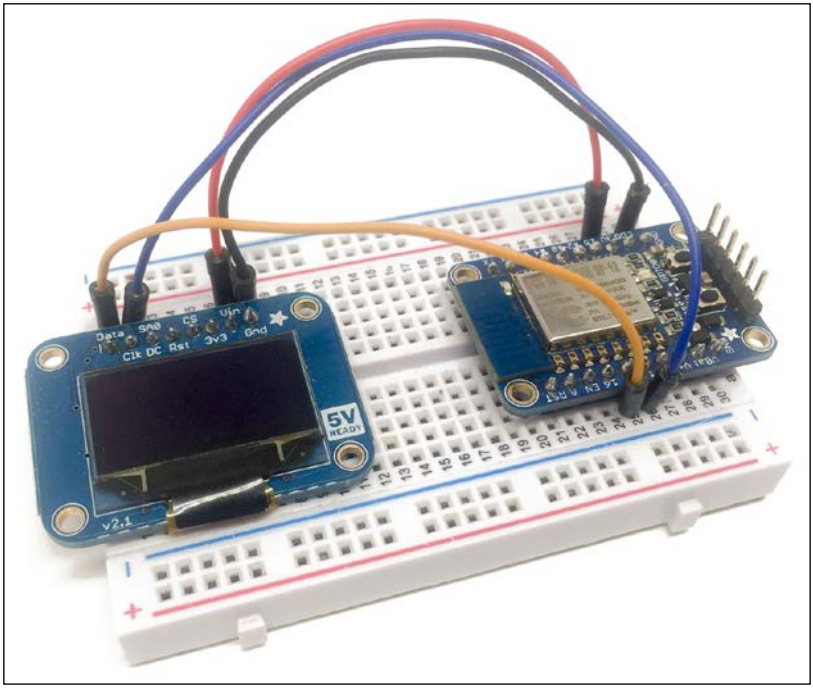
Теперь соберем проект:

1. Сначала поместите модуль ESP8266 на макетную плату вместе с OLED-экраном.
2. Затем подключите питание к OLED-экрану: подключите V_{IN} к контакту 3.3V ESP8266, а GND к GND.
3. Далее надо подключить контакты I2C дисплея к ESP8266. Подключите контакт DATA дисплея к контакту 14 ESP8266, а контакт CLK дисплея к контакту 12 ESP8266.

Обратите внимание, что некоторые OLED-экраны можно настроить на использование SPI или I2C. Убедитесь, что ваш экран настроен на использование I2C. В зависимости от марки вашего дисплея для этого может потребоваться пайка.

4. Наконец, подключите контакт RST OLED-экрана к контакту 2 платы ESP8266. Я обнаружил, что это необходимо только в том случае, если у вас есть проблемы с экраном, но на всякий случай подключите этот контакт.

Это конечный результат:



Тестирование тикера

Теперь мы собираемся настроить проект и начнем с подробного рассмотрения кода этого проекта. Конечно, вы найдете весь код для этого проекта в папке с кодами.

Он начинается с включения необходимых библиотек:

```
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include "SSD1306.h"
```

Затем мы определяем, к какому выводу подключен OLED-экран:

```
#define SDA 14
#define SCL 12
#define I2C 0x3D
```

Также нам нужно создать экземпляр ЖК-дисплея:

```
SSD1306 display(I2C, SDA, SCL);
```

Чтобы узнать текущую цену биткойнов, нам нужно использовать Coindesk API. Следовательно, мы должны определить URL-адрес API в коде:

```
const char* host = "api.coindesk.com";
```

Вам также необходимо изменить код, чтобы установить имя и пароль Wi-Fi:

```
const char* ssid     = "wifi-network";
const char* password = "wif-password";
```

В функции скетча setup () мы инициализируем

```
display.init();
display.flipScreenVertically();
display.clear();
display.display();
```

По-прежнему в той же функции мы подключаем плату к вашей сети Wi-Fi:

```
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

В функции скетча loop () мы сначала подключаемся к серверу API:

```
WiFiClient client;
constinthttpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
  }
```

Затем мы подготавливаем URL-адрес, который мы будем вызывать на сервере. Как мы видели ранее в этой главе, мы вызываем URL-адрес, чтобы получить текущую цену:

```
String url = "/v1/bpi/currentprice.json";
```

После этого мы фактически отправляем запрос:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
```

После отправки запроса мы читаем все, что возвращается с сервера:

```
String answer;
while(client.available()){
    String line = client.readStringUntil('\r');
answer += line;
  }

client.stop();
Serial.println();
Serial.println("closing connection");
```

Теперь, когда у нас есть ответ от сервера, пора его обработать.
Это начинается с извлечения точного ответа JSON из необработанного ответа:

```
String jsonAnswer;
intjsonIndex;

for (int i = 0; i <answer.length(); i++) {
if (answer[i] == '{') {
jsonIndex = i;
break;
    }
  }
```

Затем мы извлекаем объект JSON из ответа и сохраняем его в строке:

```
jsonAnswer = answer.substring(jsonIndex);
Serial.println();
Serial.println("JSON answer: ");
Serial.println(jsonAnswer);
jsonAnswer.trim();
```

Оттуда мы можем фактически извлечь цену биткойна в долларах США, выполнив некоторые операции со строкой, содержащей объект JSON:

```
intrateIndex = jsonAnswer.indexOf("rate_float");
   String priceString = jsonAnswer.substring(rateIndex + 12, rateIndex +
   18);


   priceString.trim();
   float price = priceString.toFloat();
```

Наконец, мы отображаем цену на последовательном мониторе в целях отладки:

```
Serial.println();
Serial.println("Bitcoin price: ");
Serial.println(price);
```

И мы также отображаем его на экране по центру и крупным шрифтом:

```
display.clear();
display.setFont(ArialMT_Plain_24);
display.drawString(26, 20, priceString);
display.display();
```

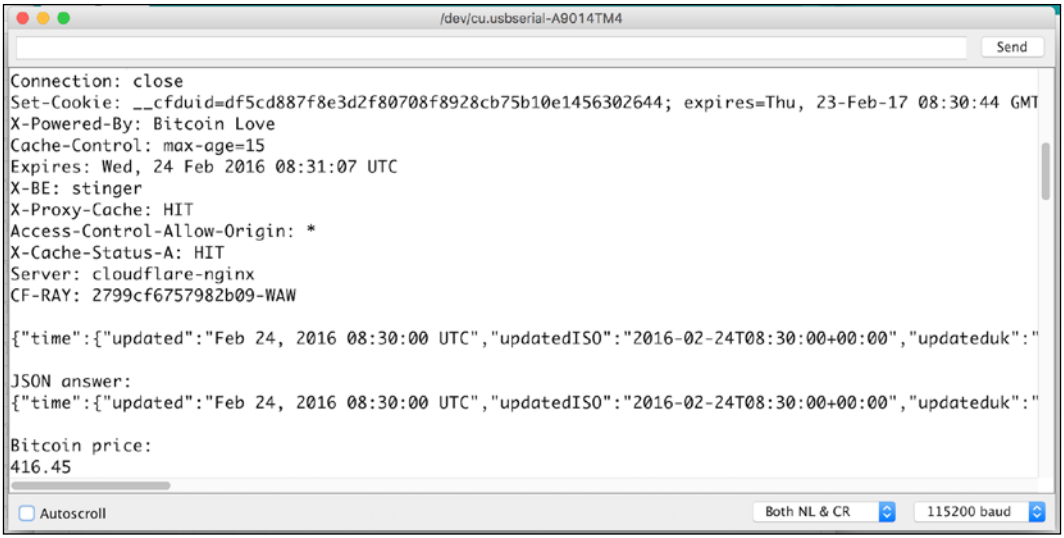
Мы также стараемся избегать постоянного повторения скетча. Вот почему вы можете настроить задержку между двумя обновлениями:

```
delay(5000);
```

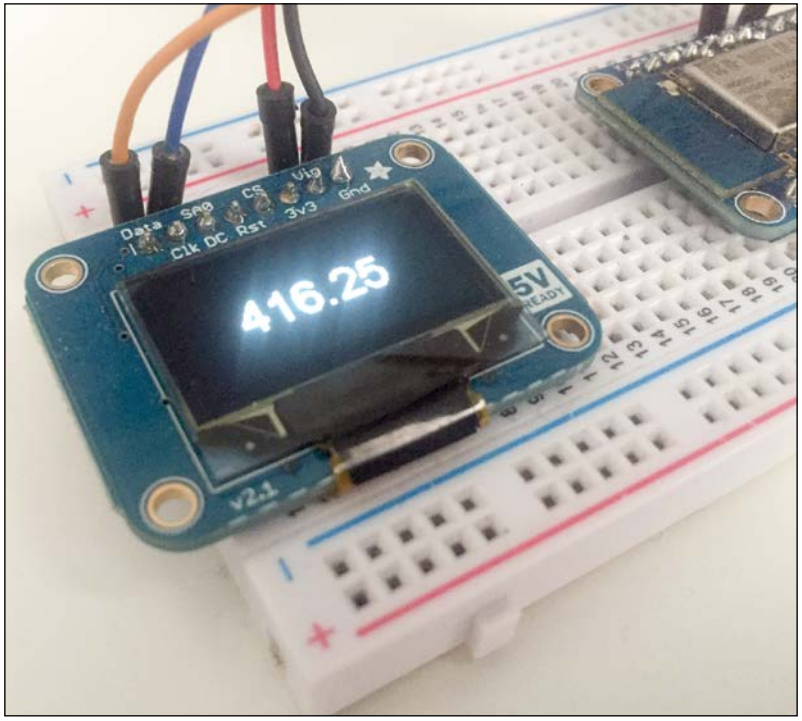
Пришло время протестировать скетч! Вы можете взять весь код из папки с кодами:

Убедившись, что вы изменили имя и пароль Wi-Fi внутри скетча, загрузите код на плату.

Как только это будет сделано, откройте последовательный монитор. Вы должны сразу увидеть, что микросхема ESP8266 получила ответ от сервера Coindesk:



Как видите, цена, когда я тестировал этот проект, составляла 416,45 USED за один биткойн. Конечно, вы также должны увидеть эту информацию на экране OLED:



Поздравляем, вы только что создали свой собственный физический биткойн-тикер!

Добавление индикаторов предупреждений в тикер

Мы сделали самый сложный проект на данный момент: мы создали физический тикер биткойнов, который показывает цену биткойна в режиме реального времени.

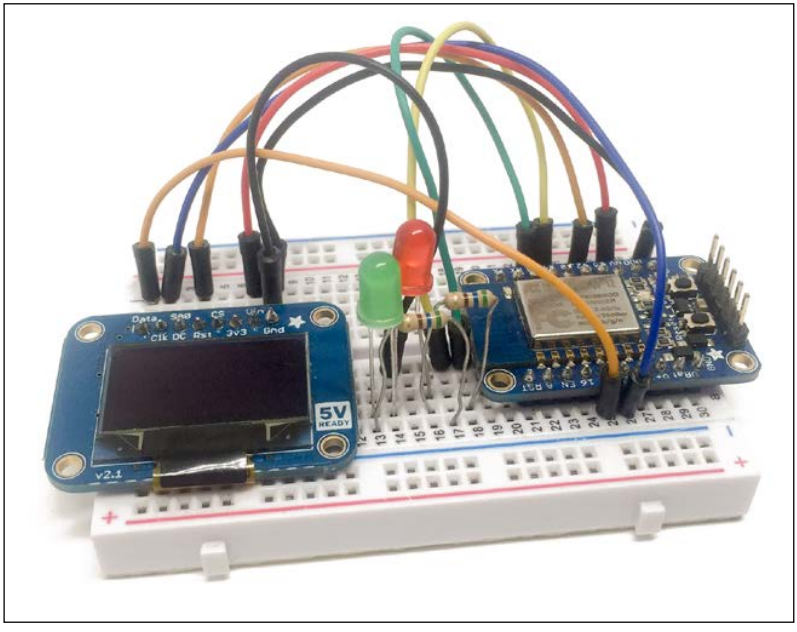
Теперь мы готовы внести в проект небольшие улучшения, чтобы сделать его еще лучше.

Например, мы собираемся добавить два светодиода, один красный и один зеленый, чтобы указать, идет ли цена биткойна вверх или вниз.

Мы будем мигать красным светодиодом, когда цена снижается, и зеленым, когда она растет.

Но сначала нам нужно добавить в проект компоненты. Просто разместите светодиоды последовательно с резисторами 330 Ом на макете, а затем подключите их, как мы видели в предыдущих главах. Убедитесь, что красный светодиод подключен к контакту 5, а зеленый светодиод - к контакту 4 платы ESP8266.

Это окончательный результат этой части:



Давайте теперь посмотрим, как настроить этот проект. Я только подробно опишу здесь, что изменилось по сравнению с предыдущим проектом:

1. Определяем контакты, к которым подключены светодиоды:

```
#define LED_PIN_UP 4
#define LED_PIN_DOWN 5
```

2. Определяем переменную для хранения предыдущего значения цены, а также пороговое значение (в долларах США), на котором мы замигаем светодиодом:

```
floatpreviousValue = 0.0;
float threshold = 0.05;
```

3. В функции скетча setup () устанавливаем выводы светодиода как выходы:

```
pinMode(LED_PIN_DOWN, OUTPUT);
pinMode(LED_PIN_UP, OUTPUT);
```

4. Внутри функции loop () мы сначала проверяем, запускаем ли код в первый раз. Если да, мы устанавливаем предыдущее значение для текущей цены биткойна:

```
if (previousValue == 0.0) {
previousValue = price;
  }
```

5. Затем мы проверяем, снизилась ли цена хотя бы на пороговую величину. В этом случае загорается красный светодиод:

```
if (price < (previousValue - threshold)) {

    // Flash LED
digitalWrite(LED_PIN_DOWN, HIGH);
delay(100);
digitalWrite(LED_PIN_DOWN, LOW);
delay(100);
digitalWrite(LED_PIN_DOWN, HIGH);
delay(100);
digitalWrite(LED_PIN_DOWN, LOW);

  }
```

6. То же самое и с зеленым светодиодом, когда цена растет:

```
if (price > (previousValue + threshold)) {

    // Flash LED
digitalWrite(LED_PIN_UP, HIGH);
delay(100);
digitalWrite(LED_PIN_UP, LOW);
delay(100);
digitalWrite(LED_PIN_UP, HIGH);
delay(100);
digitalWrite(LED_PIN_UP, LOW);

  }
```

7. Наконец, мы устанавливаем предыдущую цену на текущую цену:

```
previousValue = price;
```

Опять попробуем проект. Вы найдете код в папке с кодами. Загрузите код на плату. Вы должны сразу увидеть, что всякий раз, когда цена поднимается или падает (за пределами пороговых значений), один из светодиодов должен быстро мигать.

Резюме

Подведем итоги того, чего мы достигли в этом проекте. Мы использовали Wi-Fi-чип ESP8266, чтобы узнать цену биткойнов через веб-API, и мы отобразили эту цену на OLED-экране, подключенном к ESP8266. Затем мы также добавили в проект два светодиода, чтобы проверить, идет ли цена вверх или вниз.

Конечно, с этим проектом вы можете делать многое другое. Вы можете, например, отображать цену биткойнов в валютах, отличных от долларов США, и реализовать кнопку переключения для изменения отображаемой валюты.

В следующей главе мы увидим, как использовать ESP8266 для мониторинга и, при необходимости, полива растения (или всего сада!) Из облака.

10

Беспроводное садоводство с ESP8266

В этой главе мы снова собираемся взять все, что мы узнали до сих пор, и применить это к конкретному проекту. Здесь мы собираемся создать беспроводной облачный садоводческий проект на базе ESP8266. Это позволит вам следить за температурой и влажностью растения или всего сада и при необходимости поливать.

Сначала мы настроим проект так, чтобы он отправлял вам автоматические оповещения, когда растение высыхает. Затем мы настроим его так, чтобы вы могли не только контролировать его удаленно, но и при необходимости активировать поливочный насос. Давайте начнем!

Аппаратные и программные требования

Помимо чипа ESP8266WiFi, основным требованием этой главы является датчик температуры и влажности. Поскольку мы фактически воткнем этот датчик в землю, мы не сможем использовать обычные датчики, которые использовали ранее в этой книге.

Следовательно, нам нужно использовать датчик, который подходит для вставки в почву.
Это случай датчика, который я буду использовать в этом проекте, продаваемого Adafruit и основанного на датчике SHT10 от Sensirion:

Чтобы использовать датчик с микросхемой ESP8266WiFi, вам также понадобится резистор 10 кОм.

Вам также понадобится способ поливать растение или ваш сад, если это необходимо. Для этого я просто использовал реле 5 В, которое мы уже использовали ранее в этой главе. Таким образом, вы можете просто управлять большинством водяных насосов, которые можно найти на рынке.

Как обычно, вам также понадобятся макетная плата и перемычки.

Это список всех компонентов, которые будут использоваться в этой главе:

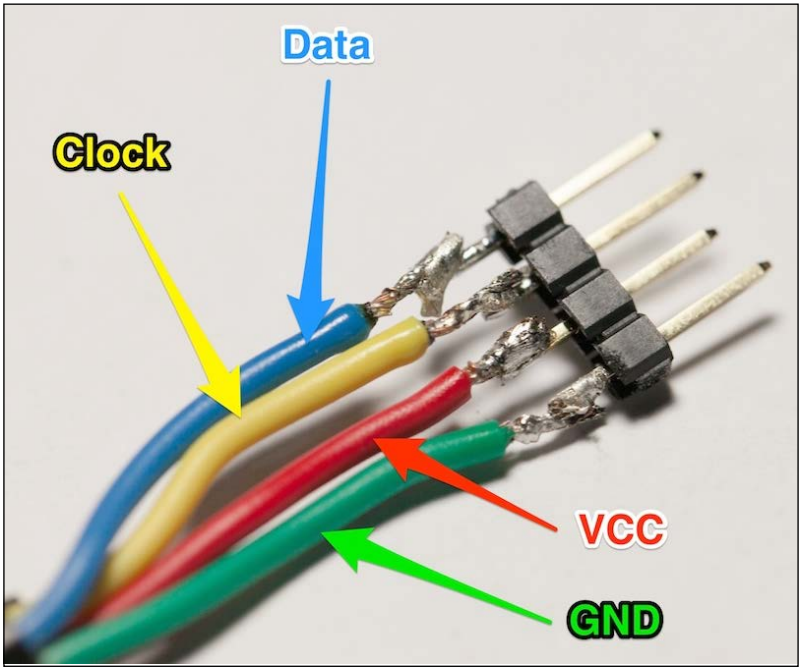
- Модуль Adafruit ES8266
- USB-модуль FTDI
- Датчик влажности почвы
- Модуль реле 5V
- Резистор 10 кОм
- Макетная плата
- Перемычки

Что касается программы, вам понадобится библиотека для датчика SHT10, которую вы можете скачать отсюда:

```
https://github.com/practicalarduino/SHT1x
```

Аппаратная конфигурация

Теперь мы собираемся собрать различные части этого проекта. Во-первых, вам нужно ознакомиться с различными контактами датчика SHT10:

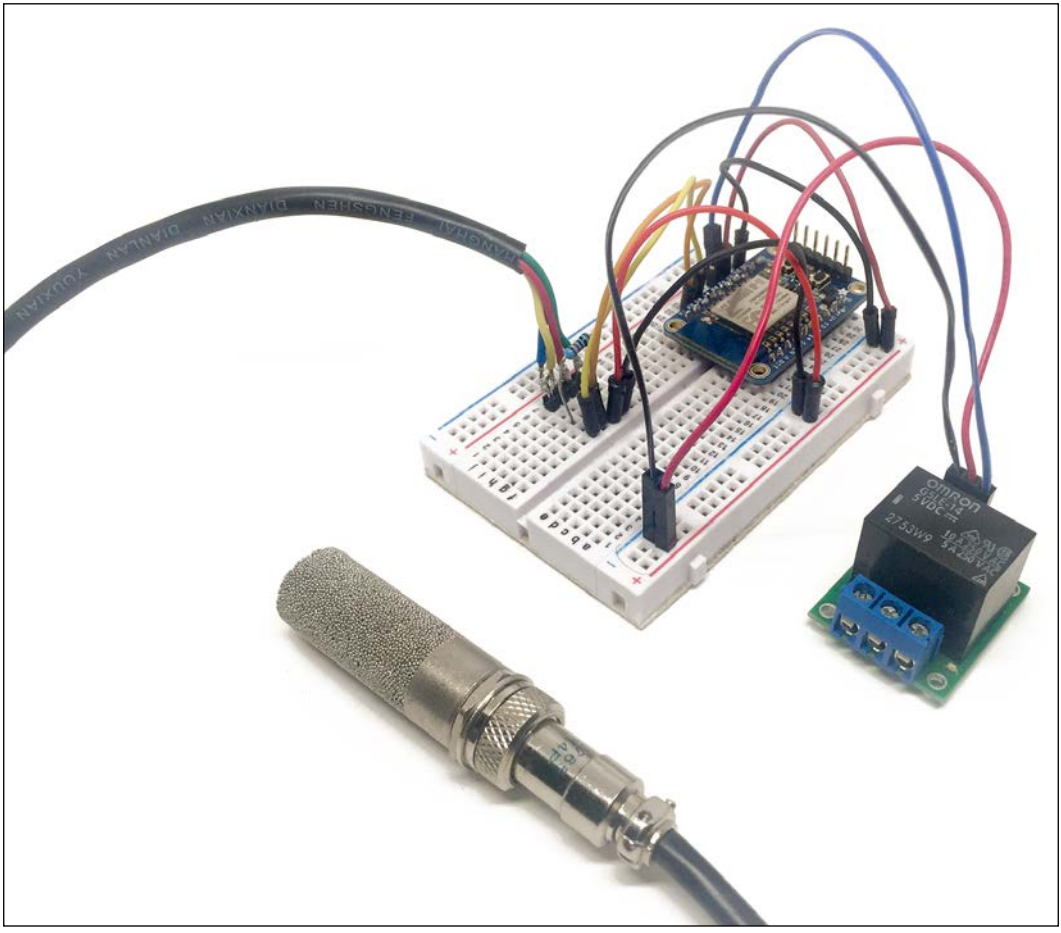


Как только это будет сделано, поместите разъем датчика на макетную плату. Также подключите контакты VCC и GND ESP8266 к красной и синей линиям питания макетной платы.

Затем подключите контакты VCC и GND датчика к красной и синей шинам питания соответственно. Затем подключите контакт данных к контакту № 4 ESP8266, а контакт синхронизации - к контакту № 5. Наконец, добавьте подтягивающий резистор 10 кОм между данными и контактами VCC датчика.

Для реле просто подключите GND к синей шине питания, VCC к красной и, наконец, контакт SIG к контакту номер 15 ESP8266.

Это конечный результат:



Однако мы еще не закончили; нам нужно что-то измерить! Теперь вы можете вставить датчик в почву, например, в горшок с растением внутри вашего дома или в саду снаружи. Вот как я вставил его, чтобы следить за одним из растений в моем доме:

Создание предупреждений о поливе вашего растения

Прежде чем создавать все захватывающие проекты беспроводного садоводства, которые вы найдете в этой главе, мы собираемся начать с одной важной вещи: проверки правильности работы датчика!

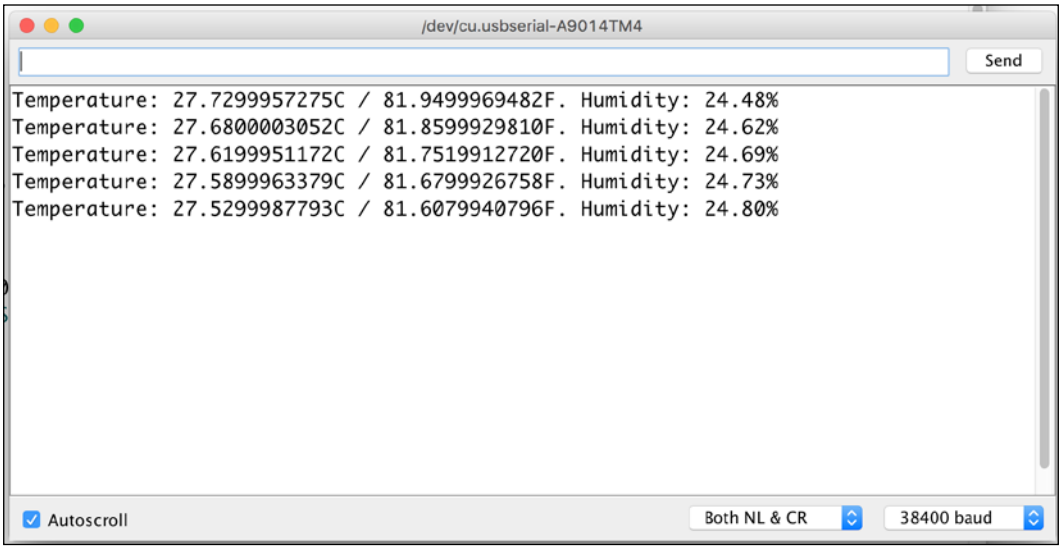
Для этого вот скетч, чтобы проверить правильность работы датчика:

```
// Library
#include <SHT1x.h>

// Pins
#define dataPin  4
#define clockPin 5

// Create instance for the sensor
SHT1xsht1x(dataPin, clockPin);
```

```
void setup()
{
Serial.begin(115200); // Open serial connection to report values to
host
Serial.println("Starting up");
}

void loop()
{
  // Variables
float temp_c;
float temp_f;
float humidity;

  // Read values from the sensor
temp_c = sht1x.readTemperatureC();
temp_f = sht1x.readTemperatureF();
humidity = sht1x.readHumidity();

  // Print the values to the serial port
Serial.print("Temperature: ");
Serial.print(temp_c, DEC);
Serial.print("C / ");
Serial.print(temp_f, DEC);
Serial.print("F. Humidity: ");
Serial.print(humidity);
Serial.println("%");

  // Wait 2 seconds
delay(2000);
}
```

Этот скетч довольно прост: мы создаем экземпляр библиотеки SHT, а в функции loop () просто проводим измерения и выводим результаты на последовательный монитор.

Просто скопируйте этот скетч в Arduino IDE, переведите ESP8266 в режим загрузчика и загрузите код на плату. Затем откройте последовательный монитор. Вот что вы должны увидеть внутри последовательного монитора:



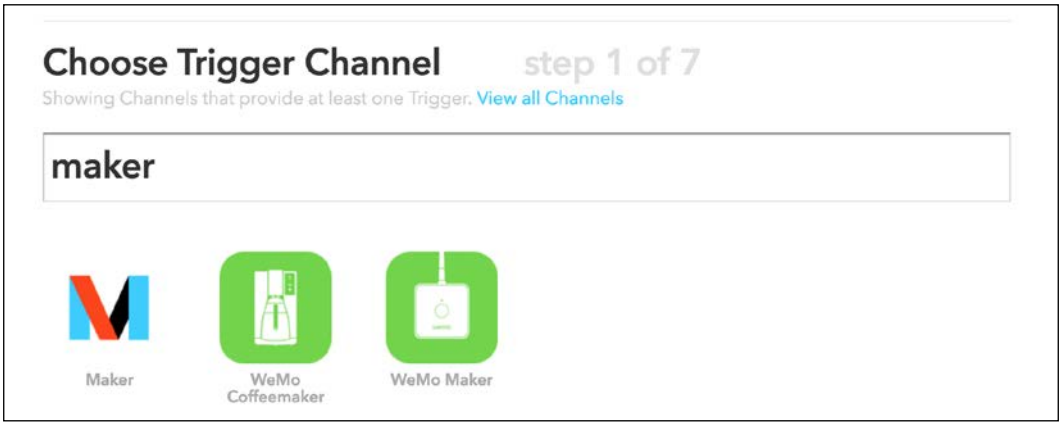
Если у вас есть значения, которые имеют смысл (например, околокомнатная температура), это означает, что ваш датчик работает правильно. В противном случае проверьте все соединения еще раз и убедитесь, что вы вставили резистор 10 кОм между выводами данных и выводами VCC датчика. На мой взгляд, это то, из-за чего датчик не работал во время первой попытки.

Теперь мы собираемся настроить автоматические оповещения на нашем телефоне, когда растение становится слишком сухим. Первый шаг для этого - создать учетную запись в IFTTT, которую мы уже использовали ранее в книге:

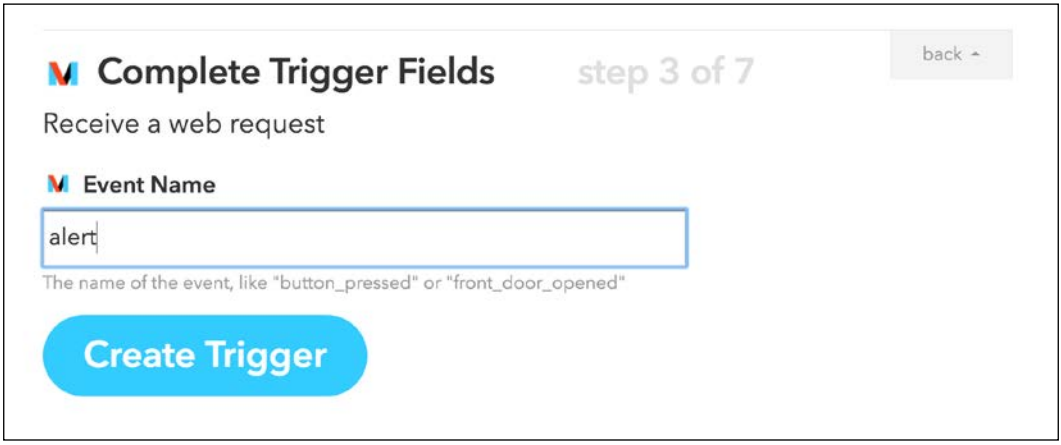
```
https://ifttt.com/
```

Я также предполагаю, что у вас уже есть каналы Maker и Pushover. Если нет, обратитесь к главе 6 «Межмашинная связь», в которой мы уже использовали IFTTT.

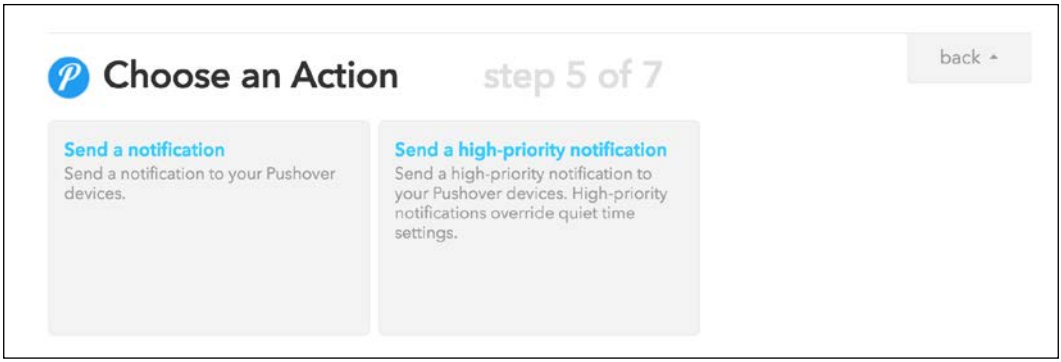
Затем создайте новый предписание с каналом Maker:



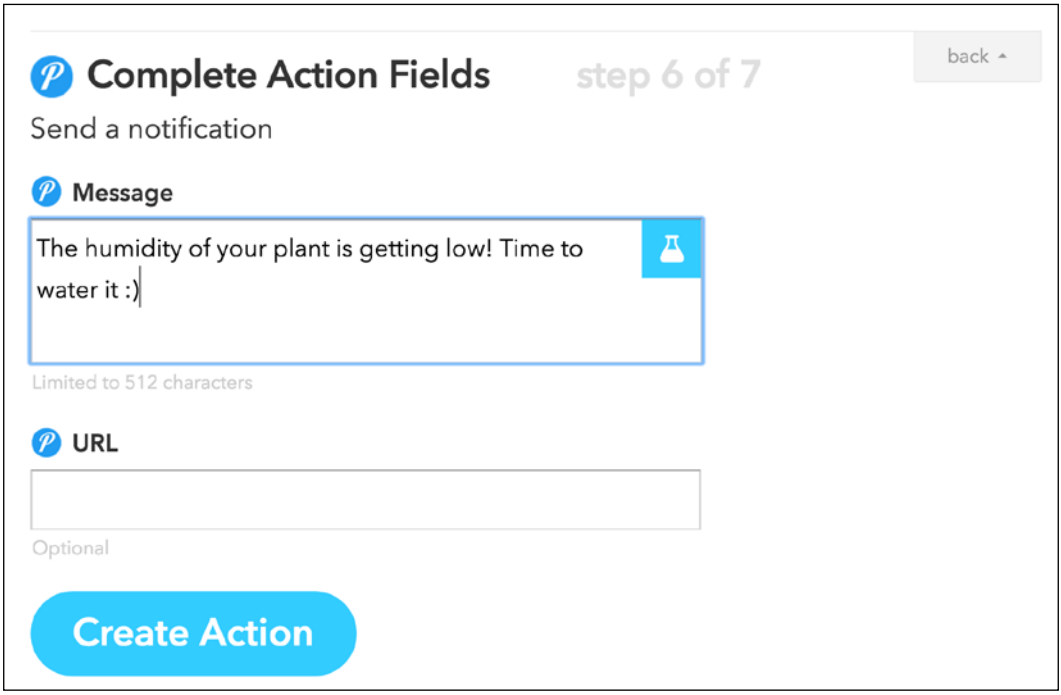
Вам нужно вставить alert (предупреждение) в качестве имени события:



Для целевого канала мы будем использовать Pushover для получения уведомлений на вашем мобильном устройстве:



Теперь вставьте сообщение в поле сообщения уведомления:



После того, как предписание создано, нам нужно настроить плату так, чтобы она автоматически отправляла предупреждения, когда влажность растения становится слишком низкой. Поскольку код здесь довольно сложный, я выделю только самые важные моменты. Он начинается с включения соответствующих библиотек:

```
#include <ESP8266WiFi.h>
#include <SHT1x.h>
```

Затем нам нужно определить ваше имя и пароль Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

Также мы определяем порог влажности, ниже которого проект автоматически отправит оповещение:

```
float threshold = 30.00;
```

Затем мы определяем параметры IFTTT. Здесь вам нужно вставить ключ вашего канала IFTTT Maker:

```
const char* host = "maker.ifttt.com";
const char* eventName   = "alert";
const char* key = "ifttt-key";
```

В функции скетча loop () после измерения с датчика мы проверяем, не ниже ли влажность порогового значения:

```
if (humidity < threshold) {
```

В этом случае мы готовим запрос, который отправим на сервер IFTTT:

```
String url = "/trigger/";
url += eventName;
url += "/with/key/";
url += key;
```

Затем мы отправили этот запрос:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                "Host: " + host + "\r\n" +
                "Connection: close\r\n\r\n");
int timeout = millis() + 5000;
while (client.available() == 0) {
if (timeout - millis() < 0) {
Serial.println(">>> Client Timeout !");
client.stop();
return;
      }
    }
```

Если запрос отправлен, мы также долго ждем до следующего оповещения, чтобы не получать оповещения от проекта постоянно:

```
delay(10 * 60 * 1000);
```

Пришло время протестировать этот первый проект главы! Обязательно используйте полный код из папки с кодами и измените учетные данные, такие как настройки IFTTT. Затем загрузите код на доску. Если влажность действительно ниже порогового значения, вы вскоре получите уведомление на свой телефон о том, что растение необходимо полить:

Контроль температуры и влажности

Во втором проекте главы мы сделаем кое-что другое. Мы будем отслеживать температуру и влажность растения с облачной панели инструментов, которую мы уже использовали ранее в этой главе. Сначала мы настроим плату, а затем настроим облачную панель управления.

Поскольку мы уже видели, как использовать облачную платформу aREST ранее в книге, я выделю только самые важные части кода.

Первый шаг - включить все необходимые библиотеки, включая библиотеку aREST:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
#include <SHT1x.h>
```

Затем мы выдаем идентификатор вашему устройству:

```
char* device_id = "gveie2y5";
```

Как обычно, вам также необходимо указать здесь свое имя Wi-Fi и пароль:

```
const char* ssid = "your-wifi";
const char* password = "your-password";
```

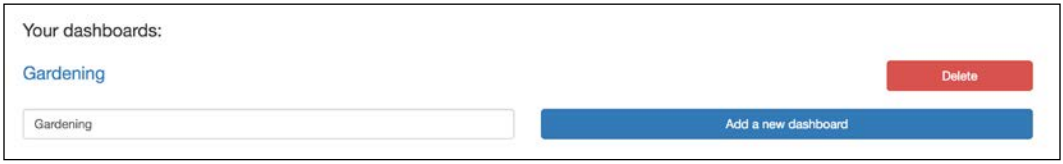
В функции скетча loop () мы проводим измерения с датчика и обрабатываем соединение с облачной платформой aREST:

```
// Read values from the sensor
temperature = sht1x.readTemperatureC();
humidity = sht1x.readHumidity();
// Connect to the cloud
rest.handle(client);
```

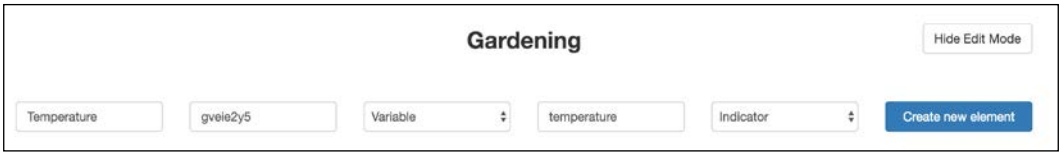
Пришло время настроить плату. Возьмите весь код из папки с кодами, а затем обязательно измените код, указав свои учетные данные и идентификатор устройства. Затем перейдите по адресу:

```
http://dashboard.arest.io/
```

Если это еще не сделано, создайте там аккаунт , а затем новую панель:



Внутри этой недавно созданной панели инструментов создайте новый индикатор переменной со следующими параметрами:



Вы должны сразу увидеть "живое" измерение, исходящее от платы:

Затем повторите тот же шаг для измерения влажности:

Теперь вы можете контролировать температуру и влажность вашего растения или сада из любой точки мира.

Но нам все еще не хватает одного ключевого элемента: насоса. Разве не было бы замечательно иметь возможность дистанционно активировать водяной насос, когда вы видите, что влажность падает? Или даже автоматически активировать его при слишком низкой влажности?

Это то, что мы увидим в следующей части этой главы.

Автоматизация вашего садоводства

Теперь мы собираемся настроить наш проект так, чтобы он автоматически поливал растение, если влажность упадет ниже заданного порога.

Первым шагом является определение двух пороговых значений:

```
floatlowThreshold = 20.00;
floathighThreshold = 25.00;
```

Здесь нам нужны два порога, потому что, если мы определим один порог, насос будет постоянно переключаться между включенным и выключенным состояниями.

Таким образом, насос будет включаться, когда влажность опустится ниже нижнего порога, и выключаться, когда мы снова достигнем верхнего порога.

Далее мы определяем, к какому выводу подключено реле:

```
#define relayPin 15
```

В функции скетча setup () мы устанавливаем вывод реле как выход:

```
pinMode(relayPin, OUTPUT);
```

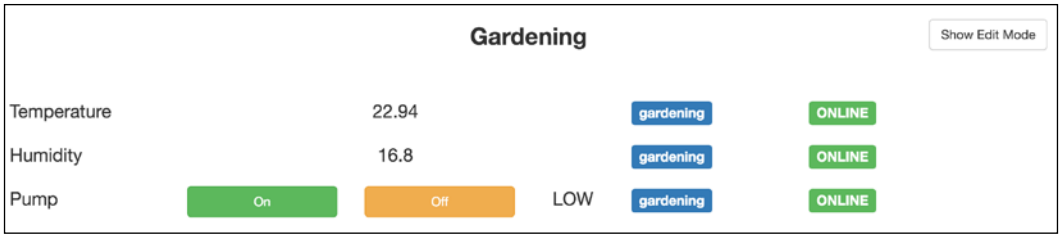
В функции loop () мы постоянно проверяем, опустилась ли влажность ниже нижнего порога или выше верхнего порога:

```
if (humidity <lowThreshold) {

// Activate pump
digitalWrite(relayPin, HIGH);

}

if (humidity >highThreshold) {

// Deactivate pump
digitalWrite(relayPin, LOW);

}
```

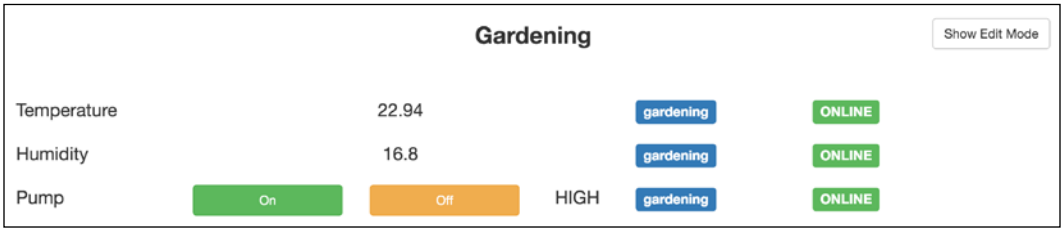
Я выделил здесь только самые важные части кода, но вы, конечно, можете получить полный код из папки с кодами. Теперь получите код и измените его своими учетными данными. Затем загрузите код на плату.

Как только это будет сделано, вернитесь к панели инструментов, которую вы создали ранее. Теперь вы можете добавить еще один элемент для насоса со следующим набором параметров:

Вы должны сразу увидеть, как появляются кнопки с текущим состоянием реле:



Теперь вы можете попробовать; нажмите кнопку Вкл (On), и помпа должна сразу включиться:



Вы также должны заметить, что каждый раз, когда влажность падает ниже порогового значения, он должен немедленно активировать насос, пока влажность снова не достигнет высокого порога.

Резюме

В этой главе мы увидели, как создать проект облачного садоводства на основе чипа ESP8266WiFi. Вы узнали, как создавать предупреждения для ваших растений или сада, как отслеживать их из любого места и как обеспечить автоматическое включение водяного насоса, когда влажность становится слишком низкой.

Конечно, вы можете взять то, что вы узнали из этой главы, и пойти дальше. Например, было бы очень легко просто добавить больше единиц в проект и контролировать несколько садов одновременно в разных местах.

В следующей главе мы собираемся создать еще один проект, используя то, что мы узнали о ESP8266: полную систему домашней автоматики, которую мы будем контролировать и отслеживать из облака.

11

Облачная система домашней автоматики

В этой главе мы собираемся использовать все, что мы узнали из книги, и применить это к области домашней автоматики. Мы собираемся создать простую, но полную систему домашней автоматики, которой мы будем полностью управлять из облака, благодаря чипу ESP8266 Wi-Fi. Система будет состоять из датчика движения, датчика температуры и влажности и светорегулятора. Таким образом, он будет имитировать все основные компоненты настоящей системы домашней автоматики.

Мы собираемся построить три проекта на основе этой системы. Сначала мы увидим, как просто контролировать каждый компонент системы с помощью онлайн-панели.

Затем мы увидим, как отправлять автоматические сигналы тревоги на ваш телефон при обнаружении движения в вашем доме. Наконец, мы увидим, как автоматизировать ваш дом с помощью IFTTT и созданной нами системы. Давайте начнем!

Аппаратные и программные требования

Давайте сначала посмотрим, что нам нужно для этого проекта. В основном вам понадобятся компоненты, которые мы уже использовали в предыдущих главах, такие как модуль ESP8266, светодиод и датчик DHT11.

Единственный новый компонент здесь - это датчик движения PIR, который мы будем использовать для обнаружения движения в нашем доме. Я использовал простой 5V-совместимый датчик движения PIR, который является довольно стандартным компонентом.

Я перечислил все компоненты для этой главы на основе одного модуля датчика движения, одного регулятора яркости светодиода и одного модуля датчика. Конечно, если вы хотите использовать больше каждого модуля, это не проблема, вам просто нужно добавить в проект больше модулей ESP8266.

Это список всех компонентов, которые будут использоваться в этой главе:

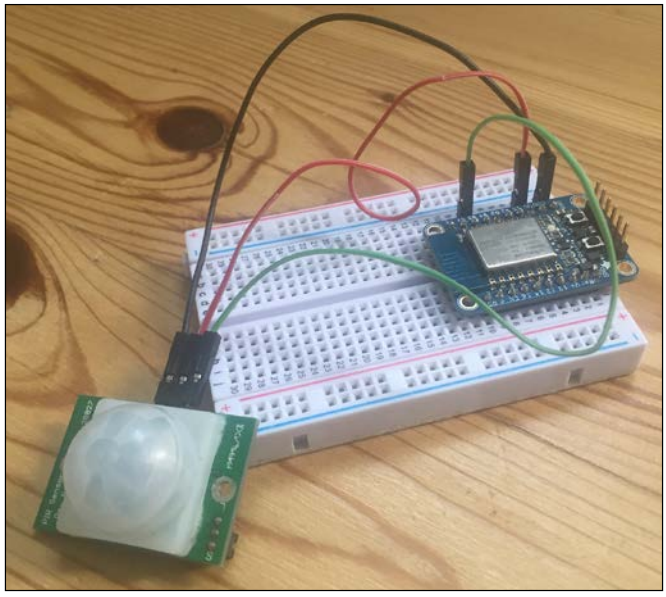
- Adafruit ES8266 module (x3)
- USB-модуль FTDI
- Датчик DHT11
- LED
- Резистор 330 Ом
- Датчик движения PIR
- Макетная плата (x3))
- Перемычки (x3)

Что касается программ, вам понадобятся библиотека aREST, библиотека PubSub, а также библиотека датчиков DHT. Мы уже установили эти библиотеки в предыдущих главах книги, но если это еще не сделано, вы можете просто установить их с помощью диспетчера библиотек Arduino IDE.

Аппаратная конфигурация

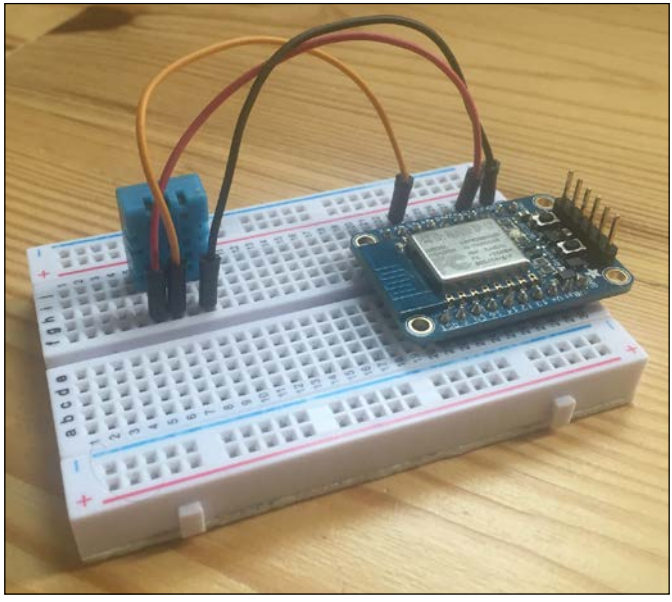
Теперь мы собираемся собрать различные части этого проекта. Сначала настроим модуль датчика движения. Для этого первого модуля, после размещения платы ESP8266 на макетной плате, подключите вывод VCC датчика к VCC, GND к GND и, наконец, вывод OUT датчика к контакту № 5 ESP8266.

Это окончательный результат этого модуля:



Теперь займемся модулем температуры и влажности. Поместите датчик на макетную плату, а затем подключите первый контакт к VCC, второй контакт к контакту № 5 платы ESP8266 и, наконец, последний контакт датчика к GND.

Это окончательный результат подключения этого модуля:

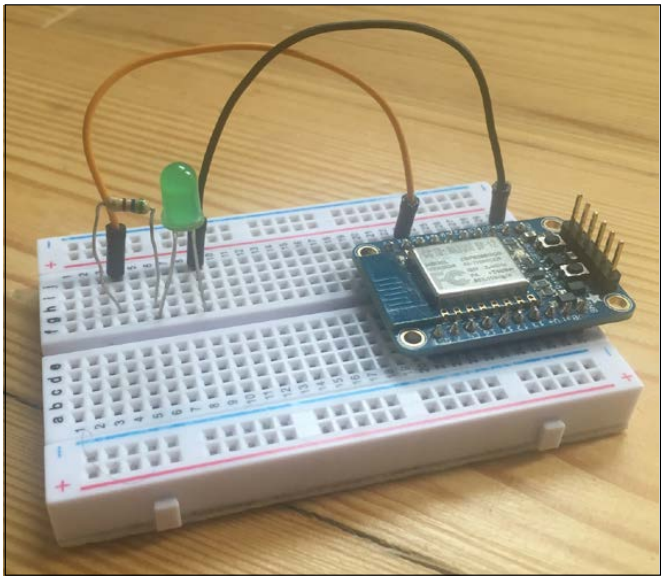


Теперь соберем модуль светодиодного диммера. Здесь мы собираемся использовать простой светодиод в качестве выхода, но вы, конечно, можете использовать его в качестве отправной точки для модуля для управления большим количеством светодиодов в вашем доме или даже лампами.

Чтобы подключить светодиод к ESP8266, просто поместите светодиод последовательно с резистором 330 Ом на макетной плате, самым длинным контактом светодиода, контактирующим с резистором.

Затем подключите другой конец резистора к контакту 5 ESP8266, а другой конец светодиода - к GND.

Это окончательный результат подключения этого модуля:



Управление своим домом с приборной панели

В первом проекте этой главы мы узнаем, как управлять всеми модулями, которые мы собрали ранее, с облачной информационной панели, используя структуру aREST, которую мы уже использовали в этой книге.

Для начала настроим все модули. Мы начнем с модуля светорегулятора, который проще всего настроить. Вот полный код этого модуля:

```
// Импортировать необходимые библиотеки
#include "ESP8266WiFi.h"#include
<PubSubClient.h>
```

```
#include <aREST.h>

// Clients
WiFiClient espClient;
PubSubClient client(espClient);

// Уникальный ID для идентификации устройства для cloud.arest.iost.io
char* device_id = "6g37g4";

// Создать экземпляр aREST
aREST rest = aREST(client);

// WiFi параметры
const char* ssid = "wifi-name";
const char* password = "wifi-pass";

// Порт для прослушивания входящих TCP-соединений
#define LISTEN_PORT           80

// Создайте экземпляр сервера
WiFiServer server(LISTEN_PORT);

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Установить обратный звонок
  client.setCallback(callback);

  //Дайте имя и ID устройству
  rest.set_id(device_id);
  rest.set_name("dimmer");

  // Подключиться к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

```
  // Запустить сервер
  server.begin();
  Serial.println("Server started");

  // Распечатать IP-адрес
  Serial.println(WiFi.localIP());

}

void loop() {

  // Обработка вызовов REST
  rest.handle(client);

}

// Обрабатывает сообщение, полученное по темам, на которые вы подписаны
void callback(char* topic, byte* payload, unsigned int length) {

  // Handle
  rest.handle_callback(client, topic, payload, length);

}
```

В этом коде нужно изменить несколько вещей. Вам необходимо заменить имя и пароль Wi-Fi на свои собственные. Вам также необходимо изменить идентификатор устройства, чтобы оно имело уникальный идентификатор ID в сети. Наконец, вы также можете изменить имя устройства, например, чтобы добавить некоторые данные о том, где находится модуль в вашем доме.

Как только это все будет сделано, загрузите код на плату, а затем перейдите к следующему устройству: датчику движения.

Для этого модуля код почти такой же, нам просто нужно добавить несколько строк, чтобы постоянно измерять состояние датчика движения и сделать его доступным в облаке. Для этого мы сначала определяем переменную, которая будет содержать состояние датчика движения:

```
int motion;
```

Затем мы выставляем эту переменную в aREST:

```
rest.variable("motion", &motion);
```

Затем внутри функции скетча loop () мы просто измеряем состояние датчика движения:

```
motion = digitalRead(5);
```

После изменения тех же параметров, что и для светодиодного модуля (учетные данные Wi-Fi, идентификатор устройства и имя), загрузите код на плату.

Наконец, займемся модулем температуры и влажности. Для этого вам нужно будет импортировать библиотеку DHT:

```
#include "DHT.h"
```

Затем вам нужно будет определить, к какому выводу подключить датчик:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

После этого создайте экземпляр датчика:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

Мы также создаем две переменные, которые будут содержать значение измеренной температуры и влажности:

```
float temperature;
float humidity;
```

В функции скетча setup () мы инициализируем датчик DHT:

```
dht.begin();
```

Мы также предоставляем эти переменные API aREST:

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Наконец, внутри функции скетча theloop () мы измеряем температуру и влажность датчика:

```
humidity = dht.readHumidity();
temperature = dht.readTemperature();
```

Опять же, измените необходимые параметры внутри скетча и загрузите его на плату.
Обратите внимание, что для питания всех модулей вы можете, например, использовать либо внешнюю батарею, либо источник питания макетной платы; Вам не нужно иметь один кабель FTDI на каждый модуль.

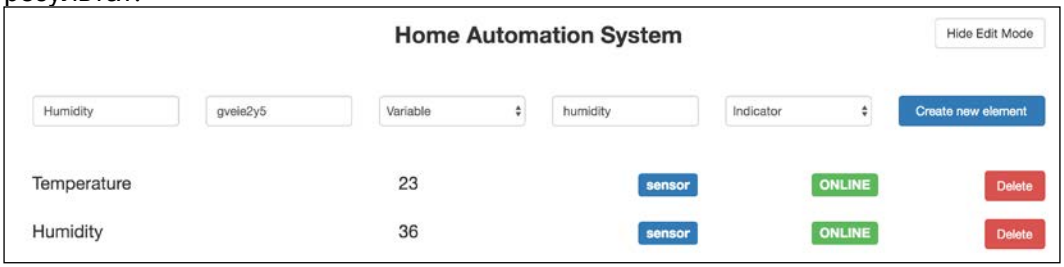
Пришло время управлять всеми нашими платами из облака! Сначала перейдите на сайт панели инструментов aREST:

```
http://dashboard.arest.io/
```

Создайте новую панель управления для вашей системы домашней автоматики:

Внутри этой панели управления переключитесь в режим редактирования и добавьте первый элемент, который будет содержать измерение температуры. Убедитесь, что вы вводите правильный идентификатор ID вашего модуля температуры и влажности:

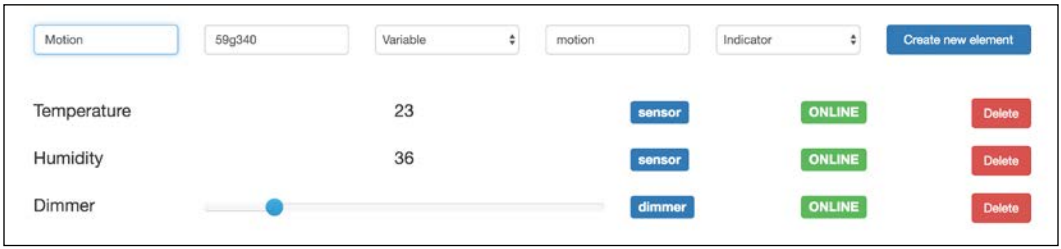
Далее проделайте то же самое с влажностью, и вы должны получить следующий результат:



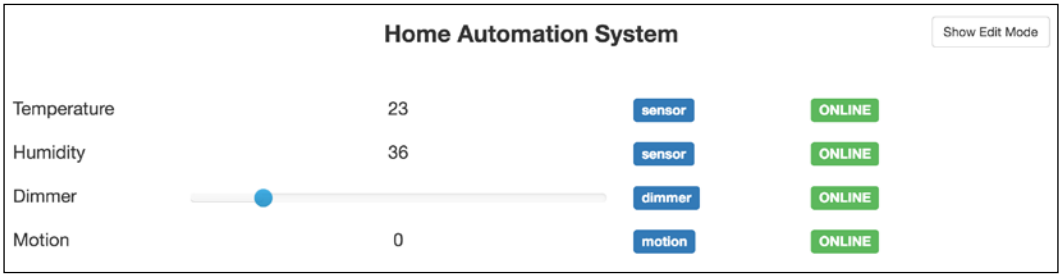
Теперь мы собираемся добавить модуль светорегулятора. Поскольку мы хотим иметь возможность управлять интенсивностью светодиодного света, создайте новый элемент с опцией Analog:

Теперь вы можете управлять яркостью светодиода с помощью ползунка на приборной панели:

Наконец, создайте новый элемент для датчика движения:



У вас должна получиться панель управления, на которой есть все элементы простой системы домашней автоматики, которую мы только что создали:



Поздравляем, вы только что создали полную систему домашней автоматики на базе ESP8266, которой теперь можно управлять из облака. Конечно, вы можете добавить в систему больше модулей и управлять ими с той же панели инструментов.

Обратите внимание, что, поскольку эта панель управления доступна из любого места, вам на самом деле не нужно находиться внутри своего дома, чтобы получить к ней доступ!

Создание облачной системы сигнализации

Теперь мы собираемся построить еще один проект на основе того же оборудования, которое мы уже создали в этой главе. На этот раз мы собираемся сделать систему сигнализации на основе облака, используя модуль ESP8266 вместе с датчиком движения PIR.

Я покажу вам, как это сделать с помощью всего одного датчика PIR, но вы, конечно, можете добавить больше модулей, которые будут разбросаны по вашему дому или любому зданию, которое вы хотите контролировать. Для этого мы снова будем использовать IFTTT, который будет отправлять вам текстовое сообщение каждый раз, когда движение обнаруживается любым из модулей датчиков движения.

Давайте сначала посмотрим, как настроить данный модуль. Поскольку это код, очень похожий на тот, который мы уже видели ранее в этой книге, я выделю здесь только самые важные части.

Вам необходимо установить ключ, связанный с вашим каналом Maker на IFTTT:

```
const char* host = "maker.ifttt.com";
const char* eventName   = "
motion_detected";const char* key = "key";
```

Затем внутри функции скетча loop () мы читаем состояние датчика движения:

```
bool motion = digitalRead(5);
```

Затем мы проверяем, было ли обнаружено движение:

```
if (motion) {
```

Если это так, мы создаем запрос, который отправим в IFTTT:

```
String url = "/trigger/";
    url += eventName;
    url += "/with/key/";
    url += key;
```

Затем мы фактически отправляем этот запрос на серверы IFTTT:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                "Host: " + host + "\r\n" +
                "Connection: close\r\n\r\n");
    int timeout = millis() + 5000;
    while (client.available() == 0) {
      if (timeout - millis() < 0) {
```

```
            Serial.println(">>> Client Timeout !");
            client.stop();
            return;
          }
        }
```

После этого читаем ответ:

```
    while(client.available()){
        String line = client.readStringUntil('\r');
        Serial.print(line);
     }
```

Затем мы долго ждем перед отправкой новых предупреждений; в противном случае мы просто отправим на ваш мобильный телефон много сообщений:

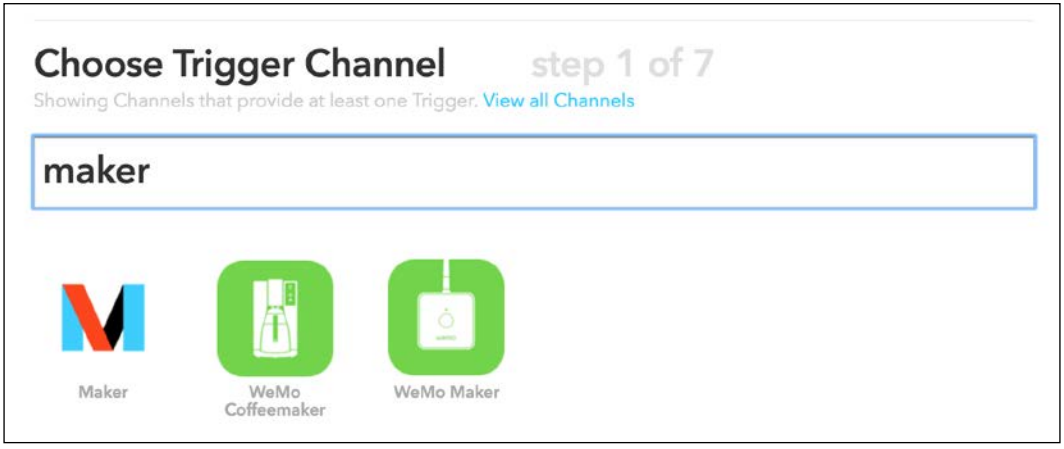
```
    delay(10 * 60 * 1000);
```

Теперь возьмите код (например, из папки с кодами), измените его своими учетными данными и загрузите на плату.

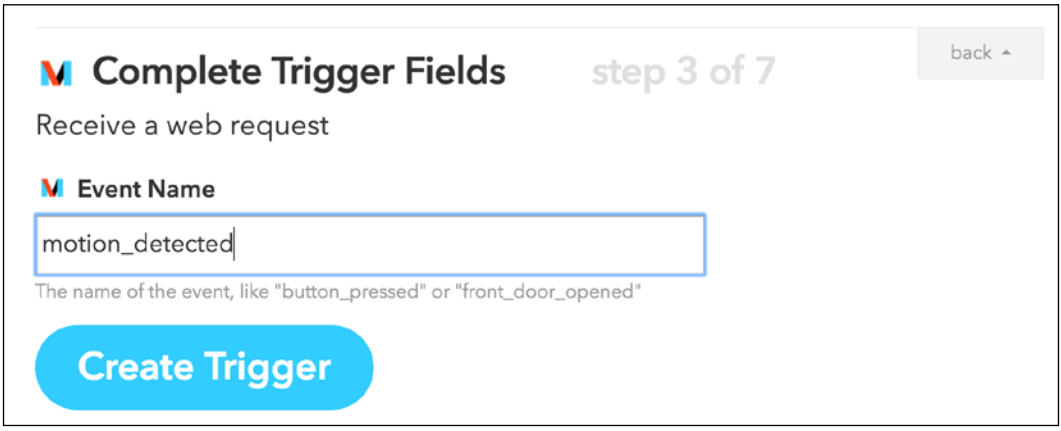
Теперь перейдите в IFTTT, чтобы создать новое предписание:

```
https://ifttt.com/
```

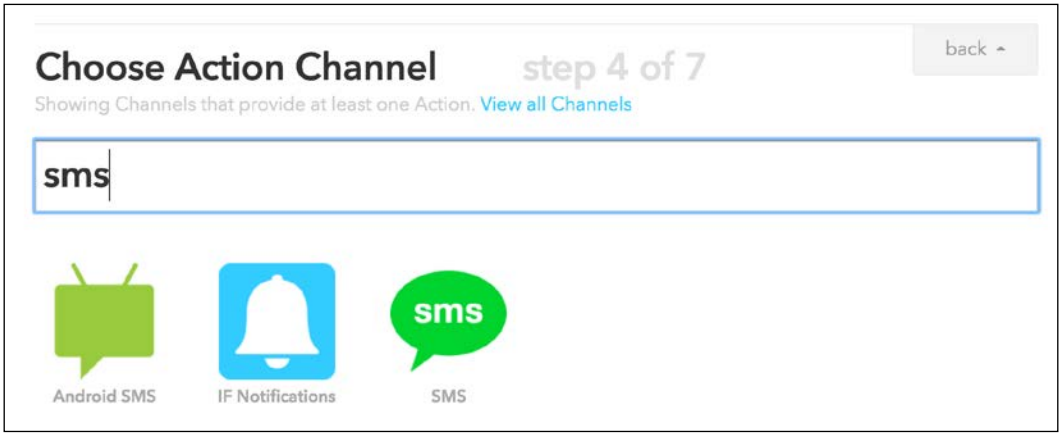
В качестве запускающего канала выберите канал Maker:



В качестве события вставьте motion_detected, что мы и поместим в код:



В качестве канала действия мы выберем здесь SMS, так как это будет самый быстрый способ связаться с вами в случае срабатывания тревоги датчиком движения:



Вы можете вставить все, что хотите, в качестве сообщения, например:

Теперь создайте предписание и активируйте его. Вы должны увидеть, что всякий раз, когда вы проводите рукой перед датчиком, он почти сразу отправляет предупреждающее сообщение на ваш телефон. Если вы хотите остановить свою систему сигнализации, это так же просто, как деактивировать рецепт на IFTTT.

Вы, конечно, можете добавить в систему больше датчиков, чтобы каждый из них публиковал одно и то же предупреждение на IFTTT. Поэтому всякий раз, когда какой-либо датчик обнаруживает движение, вы получите уведомление на свой телефон.

Автоматизация вашего дома

В последней части этой главы мы еще немного поиграем с IFTTT. На этот раз вместо датчика движениямы увидим, как использовать различные запускающие каналы IFTTT для управления светодиодным модулем. Конечно, помните, что вы можете заменить светодиод любым прибором, которым хотите управлять, например лампой.

Канал Maker в IFTTT также можно использовать в качестве канала действий, и это то, что мы собираемся использовать здесь. Мы будем использовать его для вызова aREST API всякий раз, когда срабатывает заданное условие.

Сначала мы собираемся снова настроить модуль, чтобы он мог получать команды из облака. Это часть кода перед функцией setup ():

```
// Импортировать необходимые библиотеки
#include "ESP8266WiFi.h"
#include <PubSubClient.h>
#include <aREST.h>

// Клиенты
WiFiClient espClient;
PubSubClient client(espClient);

// Уникальный ID для идентификации устройства для cloud.arest.io
char* device_id = "6g37g4";

// Создать экземпляр aREST
aREST rest = aREST(client);

// Параметры WiFi
const char* ssid = "wifi-name";
const char* password = "wifi-pass";

// порт для прослушивания входящих TCP-соединений
#define LISTEN_PORT           80

// Создайте экземпляр сервера
WiFiServer server(LISTEN_PORT);
```

Это функция кода setup ():

```
void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Установить обратный звонок
  client.setCallback(callback);

  // Дайте имя и ID устройству
  rest.set_id(device_id);
  rest.set_name("dimmer");
```

```
  // Подключиться к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Запустить сервер
  server.begin();
  Serial.println("Server started");

  // Распечатать IP-адрес
  Serial.println(WiFi.localIP());

  // Вывод светодиода на выход
  pinMode(5, OUTPUT);

}
```

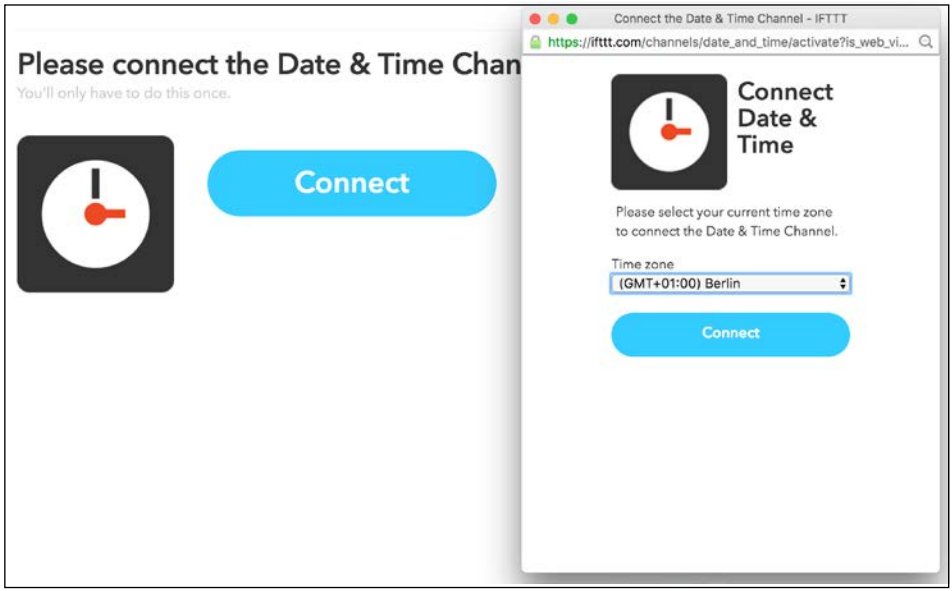
Наконец, вот функция кода loop ():

```
void loop() {

  // Обработка вызовов REST
  rest.handle(client);

}

// Обрабатывает сообщение, полученное по темам, на которые вы подписаны
void callback(char* topic, byte* payload, unsigned int length) {

  // Вручную
  rest.handle_callback(client, topic, payload, length);

}
```

Теперь измените важные учетные данные (имя и пароль Wi-Fi, а также ID устройства) в коде и загрузите их на плату.

Затем вернитесь к IFTTT. Первое, что мы собираемся сделать, это сделать проект, чтобы зажечь светодиод в заданное время (например, когда становится темно на улице), а затем выключить в другое время (например, когда вы ложитесь спать). .

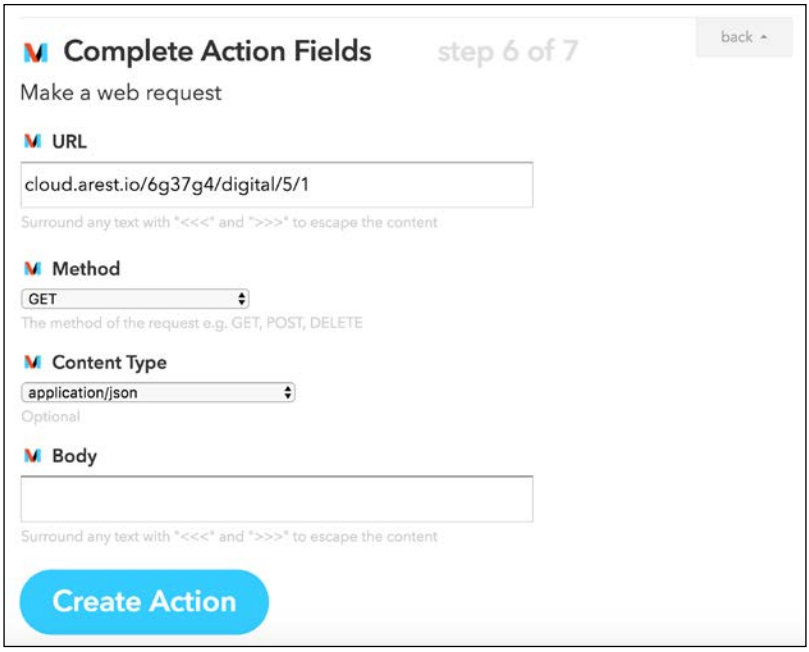
Для этого создайте новый рецепт с датой и временем - Date & Time в качестве канала. Возможно, вам придется сначала подключить его, если вы никогда раньше не использовали его:



В качестве инициатора выберите «Every day - Каждый день» и введите время, когда должен включаться светодиод.

Затем для канала действия в IFTTT выберите канал Maker, а затем выберите Make a web request - Сделать веб-запрос. Это позволит IFTTT отправить команду на облачный сервер aREST.io.

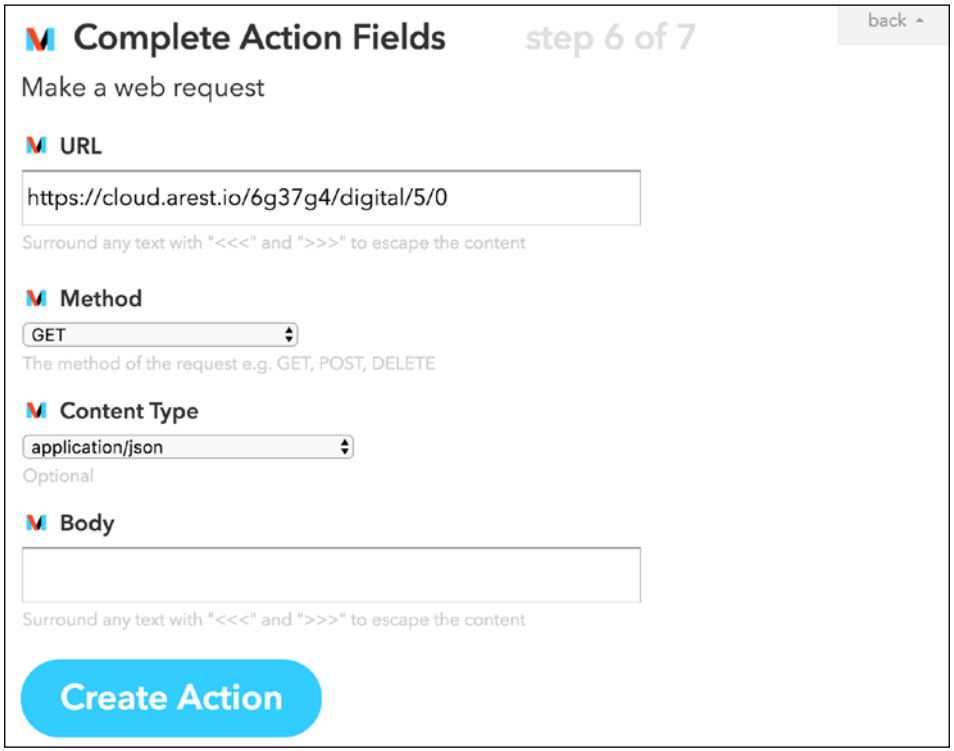
В качестве запроса введите следующие параметры, конечно же, изменив идентификатор устройства на тот, который вы использовали в скетче:



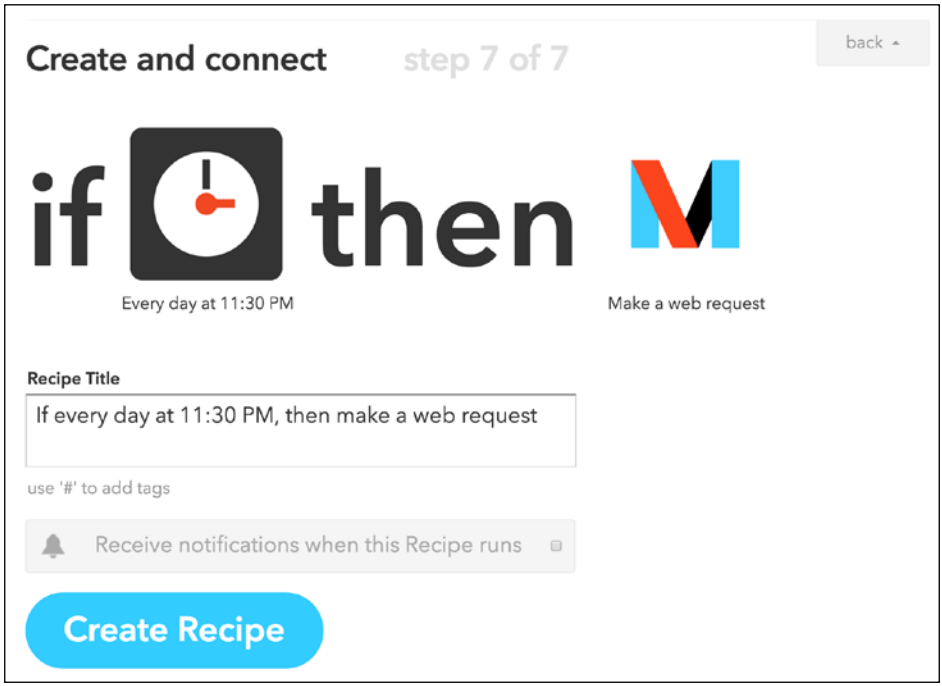
Теперь сделайте то же самое со временем, когда вы хотите, чтобы светодиод выключался, например, в 23:30:

Выберите тот же канал действий, что и раньше:

Для веб-запроса введите те же параметры, что и раньше, но на этот раз с командой переключить вывод 5 на НИЗКИЙ:



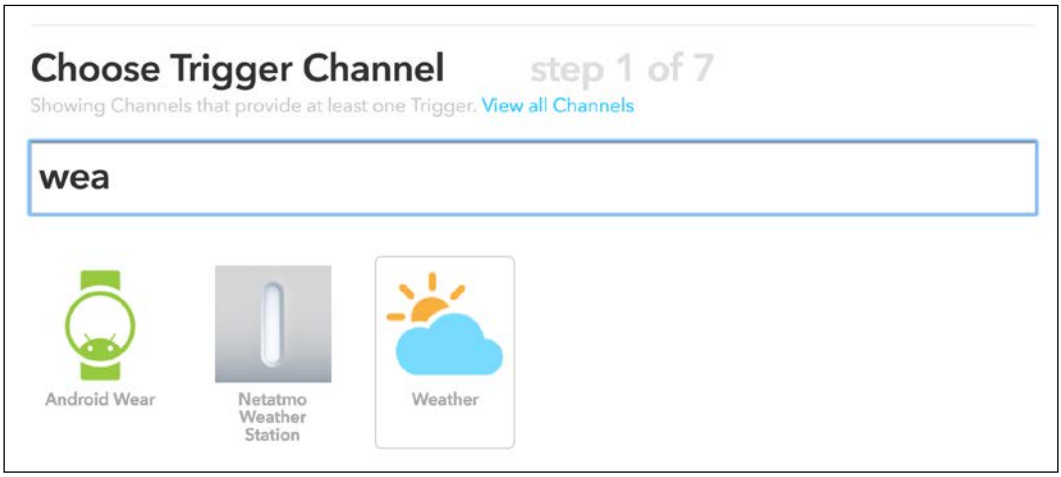
Теперь проверьте предписание:



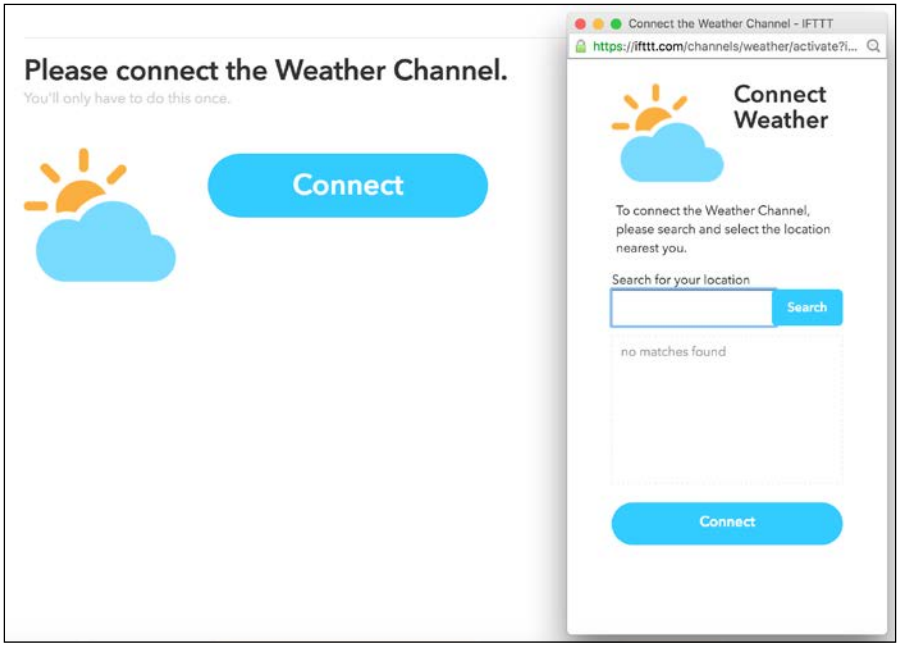
Пришло время увидеть предписание в действии! Убедитесь, что плата ESP8266 настроена правильно, а также что рецепты активированы в IFTTT. Как только условия времени соблюдены, вы должны сразу увидеть, как светодиод включается или выключается. Конечно, вы можете изменить время в рецептах, чтобы проверить их.

Теперь давайте поиграем с другим каналом запуска, чтобы увидеть, насколько мощным является IFTTT. Вы можете, например, использовать канал Weather - Погода, чтобы проверить, не наступил ли закат, чтобы автоматически включить светодиод в это время без необходимости вводить фиксированное время.

Для этого создайте новый рецепт и выберите канал Погода, к которому вам нужно подключиться:

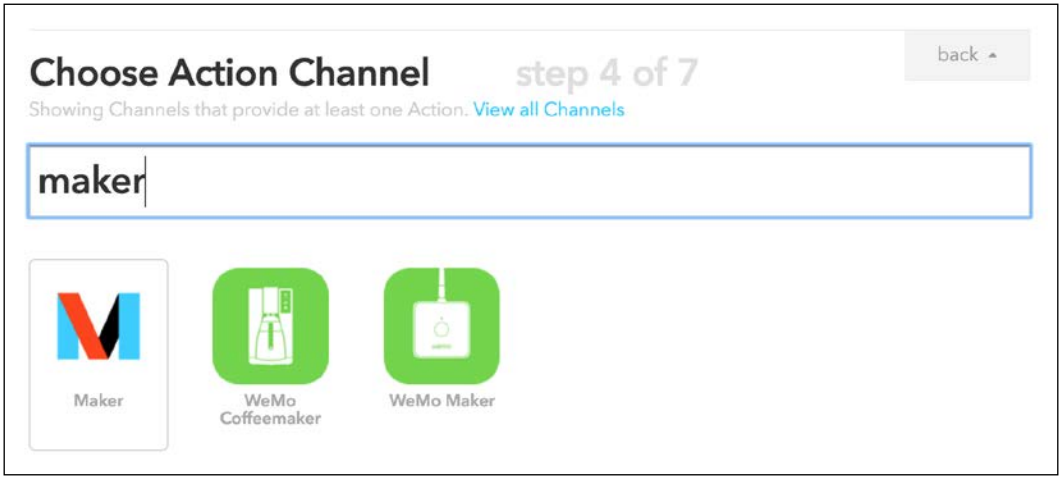


Чтобы подключить канал Weather - Погода, вам просто нужно ввести свое текущее местоположение:

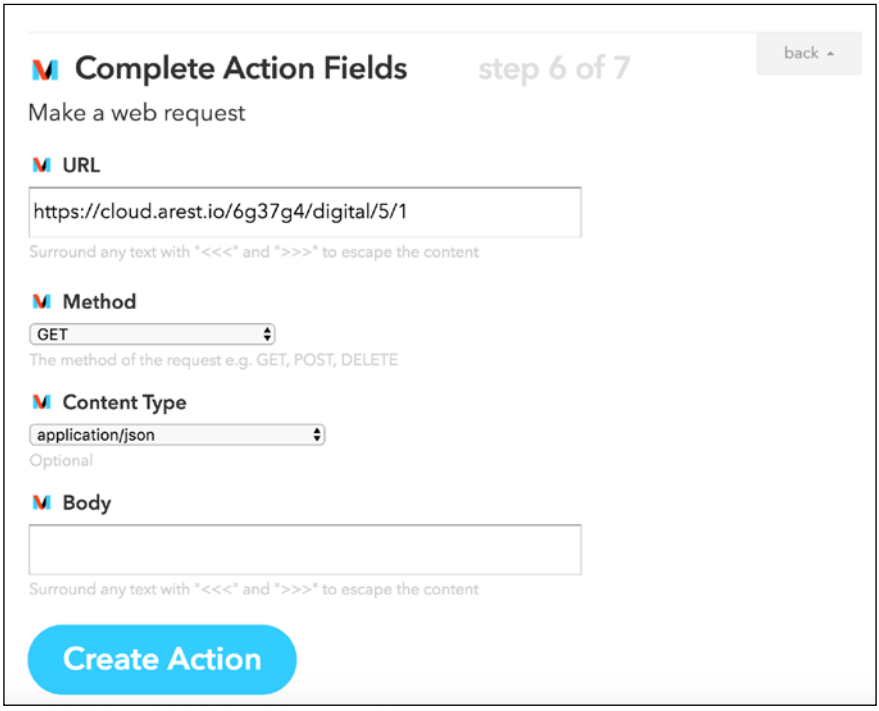


В качестве инициатора я выбрал запуск на закате:

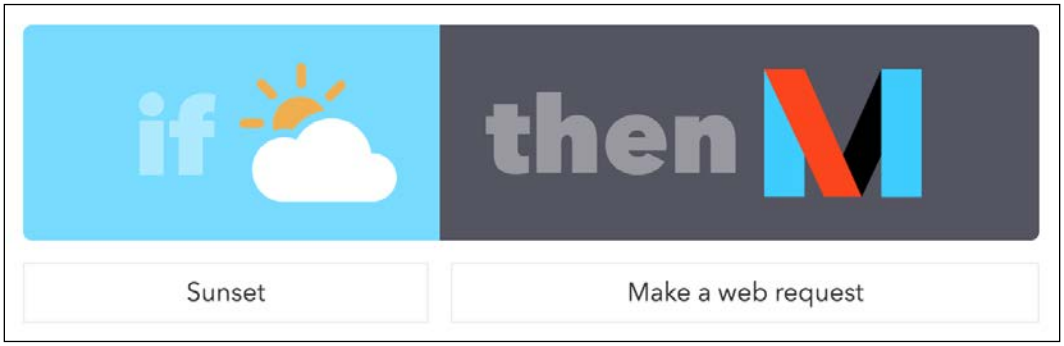
В качестве канала действия я выбрал канал Maker, как и раньше:



Кроме того, как и в предыдущем рецепте, я решил включать светодиод всякий раз, когда запускается уведомление:



Вы не можете увидеть только что созданное уведомление:



Теперь вы должны увидеть, что каждый раз, когда приближается закат, автоматически включается светодиод. Конечно, вы можете поиграть с большим количеством условий внутри **IFTTT** и даже использовать канал **Maker** как сигнализатор и канал действия, чтобы связать различные модули вашей системы домашней автоматики через **IFTTT.**

Резюме

В этой главе мы построили различные компоненты системы домашней автоматики на основе ESP8266 и увидели, как управлять всем из облака. Сначала мы создали облачную панель управления для управления всеми устройствами из единого интерфейса. Затем мы снова использовали IFTTT для создания системы сигнализации, которая автоматически отправляет вам оповещения на ваш телефон при обнаружении движения. Самый простой способ построить свою собственную систему на основе этого проекта - просто добавить дополнительные модули, чтобы они соответствовали вашим потребностям в вашем собственном доме. Например,очень легко добавить несколько модулей датчиков движения, чтобы вы могли развернуть полную систему сигнализации в своем доме и управлять ею с помощью простого веб-браузера.

В следующей главе мы собираемся использовать ESP8266 для совершенно другого приложения: создания мобильного робота, которым мы будем управлять из облака.

12

Робот ESP8266 с облачным управлением

До сих пор в этой книге мы в основном использовали чип Wi-Fi ESP8266 для создания проектов Интернета вещей, которые были связаны с домашней автоматикой или областями безопасности, такими как дверной замок с дистанционным управлением или полная система домашней автоматики, которую мы создали. Глава 11, Облачная система домашней автоматики. Однако ESP8266 - очень универсальный чип, и его можно использовать в нескольких областях, кроме двух, которые я цитировал выше. Например, его также можно использовать в качестве «мозга» мобильного робота, и именно этим мы и собираемся заниматься в этой главе.

Мы собираемся построить небольшого мобильного робота, который будет полностью управляться чипом ESP8266 Wi-Fi. Мы собираемся подключить двигатели робота к микросхеме Wi-Fi ESP8266, а затем чип будет общаться по беспроводной сети с облачной платформой, так что роботом можно будет управлять из любого места. Наконец, мы собираемся управлять роботом из облачной панели управления. Давайте начнем!

Аппаратные и программные требования

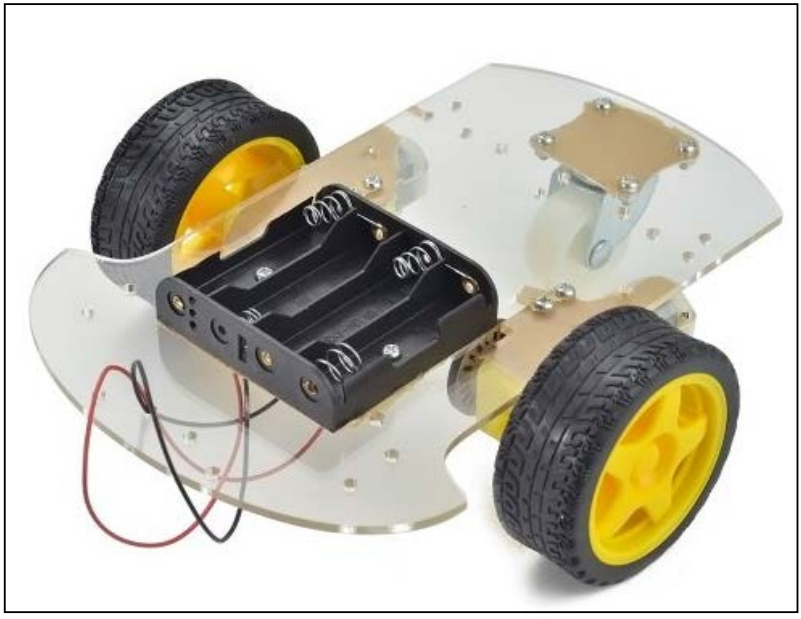
Первое, что нам понадобится для этого проекта, - это Wi-Fi-чип ESP8266. Что касается большей части этой книги, я буду использовать здесь плату Adafruit Huzzah ESP8266.

Тогда другим важным компонентом этого проекта станет платформа роботов. На рынке есть много вариантов, но на самом деле их не так много, которые предназначены для ESP8266, поэтому нам нужно будет выбрать очень общую платформу, на которую мы можем просто установить компоненты по нашему выбору.

Нам в основном нужно определенное количество функций, чтобы иметь платформу робота для этого проекта:

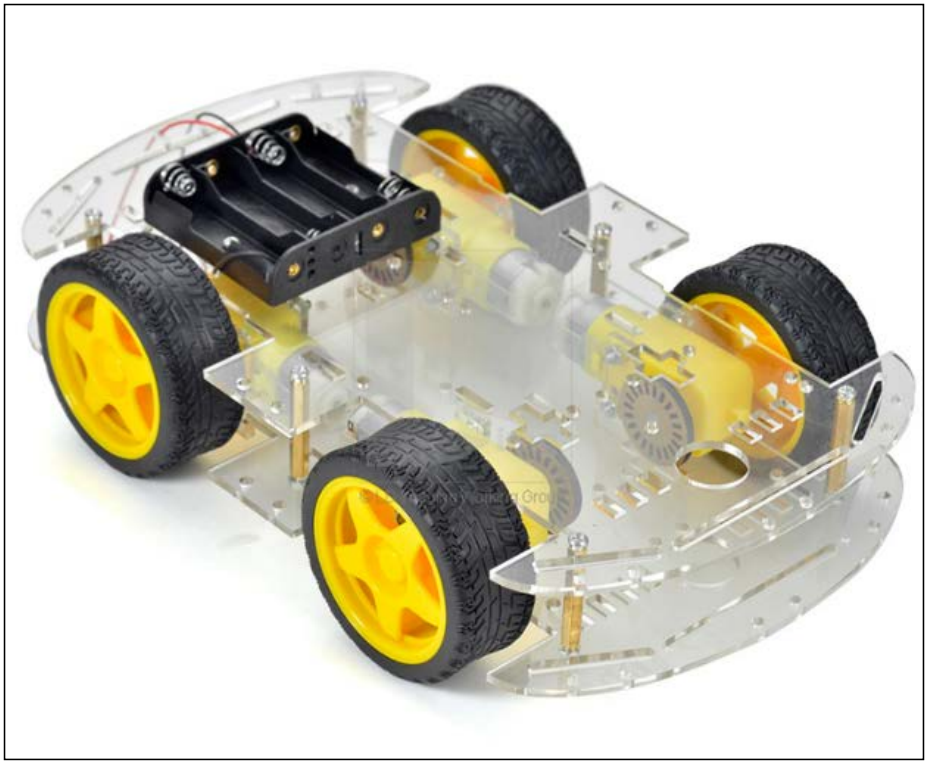
- Платформа должна иметь как минимум два колеса.
- Платформа должна иметь два мотора.
- Платформа должна быть достаточно большой, чтобы на ней можно было разместить ESP8266, макетную плату и аккумуляторы.

На самом деле существует большое количество платформ роботов, которые будут работать с этими спецификациями. Первая категория - это, конечно, простые комплекты двухколесных роботов, такие как этот комплект от EmGreat:



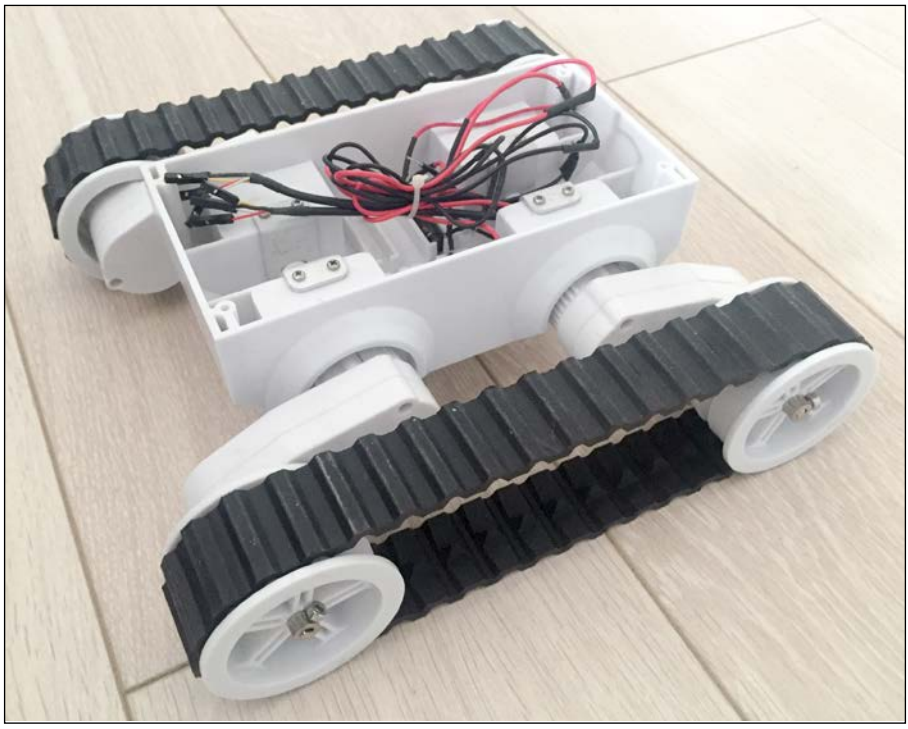
У этих платформ просто два колеса с двигателем на каждом, а затем свободное колесо спереди, чтобы робот был устойчивым. Затем вы можете просто установить на него свое оборудование.

Затем у вас есть четырехколесные роботы, которые по сути представляютсобой небольшую машину, на которую вы можете установить свое собственное оборудование. Это четырехколесный комплект той же марки:



Если вы собираетесь использовать такую платформу, убедитесь, что есть только два двигателя, так как в этой главе я опишу проект, использующий только два двигателя. Конечно, вы также можете легко адаптировать код из этой главы к платформе четырехмоторного робота.

Я сделал аналогичный выбор для этой главы, так как у меня все еще был робот-вездеход, который я хотел использовать:



Это в основном тот же робот, который я представил ранее, но на этот раз с резиновыми лентами вокруг колес, как у танка. Это гарантирует, что на робота не слишком сильно повлияет то, что стоит за ним, например некоторые неровности на земле. Также у него всего два мотора.

После этого вам потребуются дополнительные компоненты для управления роботом. Во-первых, это микросхема L293D, которая представляет собой интегральную схему, специализирующуюся на управлении двигателями. Действительно, напрямую управлять моторами с ESP8266 было бы невозможно.

Вам также понадобится аккумулятор, адаптированный к двигателям вашей роботизированной платформы. Двигатели шасси марсохода, которые я использую, были рассчитаны на 7 В, поэтому я использовал эту батарею на 7,4 В:

Также вам понадобятся обычные макетные платы и перемычки.

Это список всех компонентов, которые будут использоваться в этой главе:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- Шасси робота-вездехода с двумя моторами

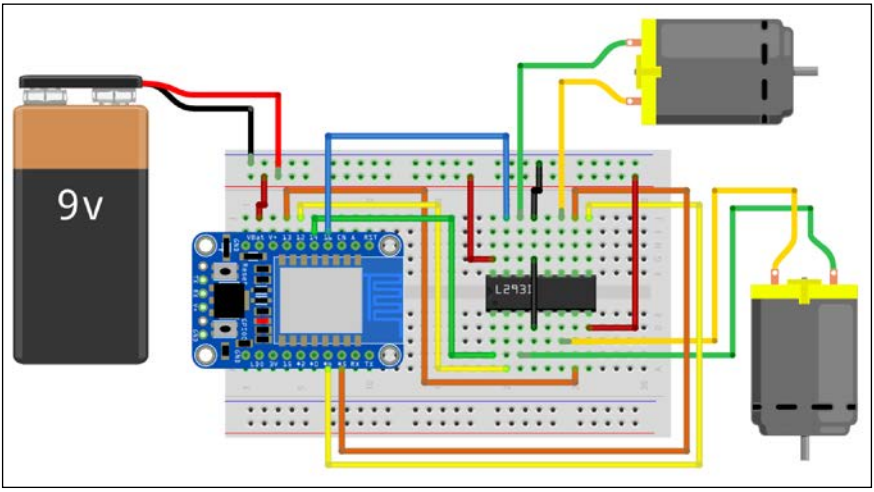
- Драйвер двигателя L293D
- Аккумулятор 7,4 В с разъемом питания

- Макетная плата
- Перемычки

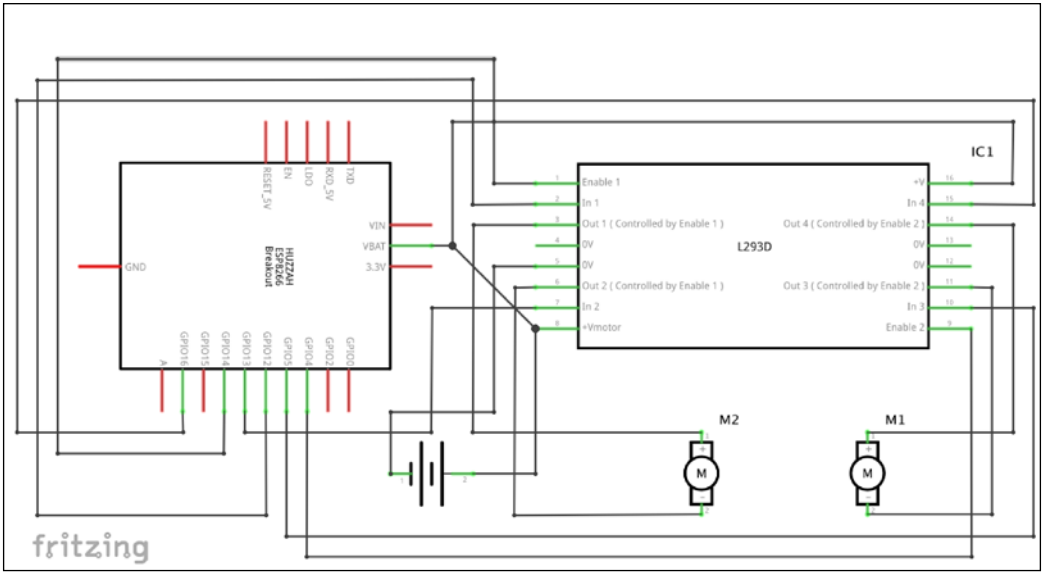
Что касается программ, вам потребуются IDE Arduino и библиотека aREST, которые мы уже использовали ранее в этой книге.

Аппаратная конфигурация

Теперь соберем проект. Поскольку это довольно сложно, я сделал подробную схему, чтобы вы могли понять как все соединять:

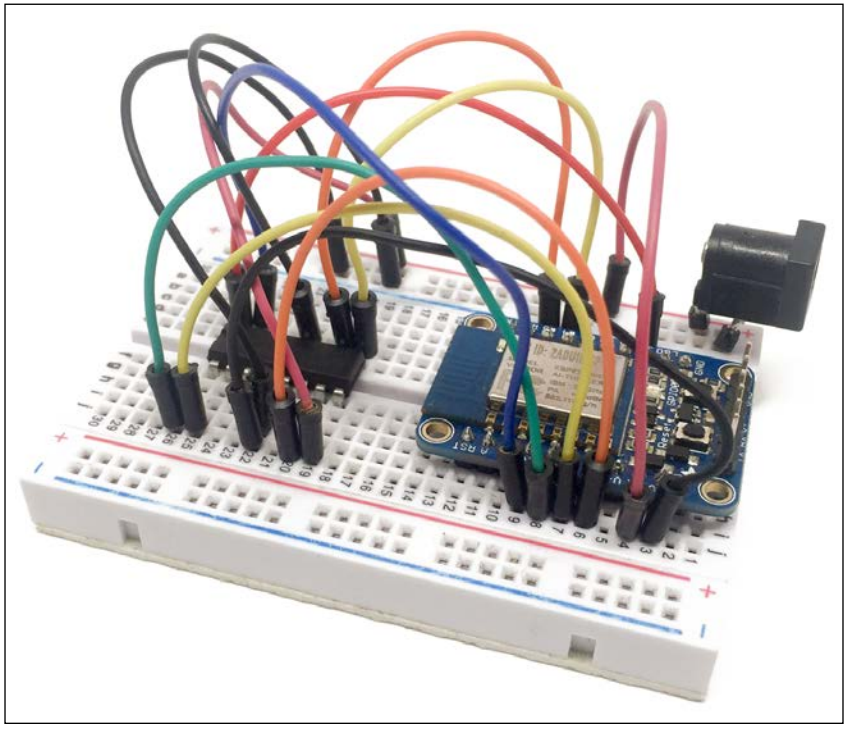


Чтобы помочь вам, здесь также представлена подробная схема:

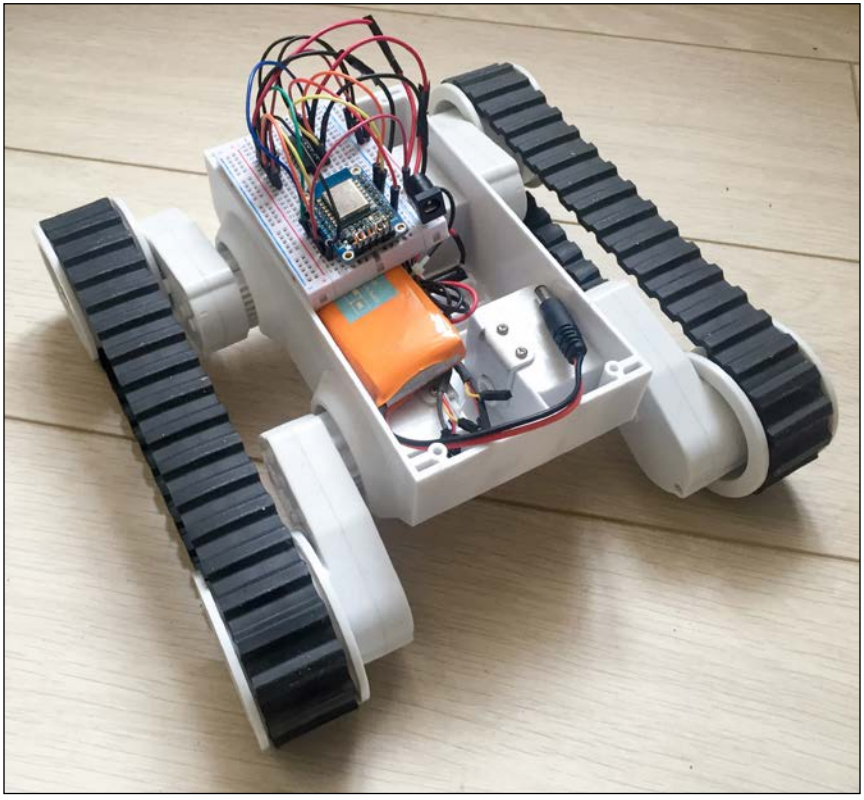


Первый шаг - собрать все компоненты на макетной плате. Только после этого поместите макет на шасси робота, а затем подключите его к двигателям. В конце подключите аккумулятор к проекту.

Это окончательный результат, показывающий только макет, без всех подключений к двигателям:



Это полностью собранная роботизированная платформа:



Обратите внимание, что платформа поставляется с крышкой, чтобы скрыть компоненты внутри, но здесь я решил оставить ее открытой, чтобы вы могли видеть внутреннюю часть проекта. Но не делайте этого с собственным роботом и не обязательно прикручивать / приклеивать все компоненты к шасси!

Тестирование моторов

Прежде чем мы начнем управлять роботом из облака, мы проведем простой тест, чтобы убедиться, что двигатели работают правильно. Это также позволит вам увидеть, как писать код для управления двигателями.

Это полный скетч этой части:

```
// Определите пины двигателя
int motorOnePlus = 12;int
motorOneMinus = 13;
int motorOneEnable = 14;

int motorTwoPlus = 5;
int motorTwoMinus = 16;
int motorTwoEnable = 4;

void setup()
{

   Serial.begin(1152000);

   // Установить pins
   pinMode(motorOnePlus, OUTPUT);
   pinMode(motorOneMinus, OUTPUT);
   pinMode(motorOneEnable, OUTPUT);

   pinMode(motorTwoPlus, OUTPUT);
   pinMode(motorTwoMinus, OUTPUT);
   pinMode(motorTwoEnable, OUTPUT);

}

void loop()
{

   // Ускоряться вперед
   setMotorOne(true, 500);
   setMotorTwo(true, 500);

   // пауза
   delay(5000);

   // Stop
   setMotorOne(true, 0);
   setMotorTwo(true, 0);
```

```
    // пауза
    delay(5000);

}

// Функция управления двигателем
void setMotorOne(boolean forward, int motor_speed){
    digitalWrite(motorOnePlus, forward);
    digitalWrite(motorOneMinus, !forward);
    analogWrite(motorOneEnable, motor_speed);
}

// Функция управления двигателем
void setMotorTwo(boolean forward, int motor_speed){
    digitalWrite(motorTwoPlus, forward);
    digitalWrite(motorTwoMinus, !forward);
    analogWrite(motorTwoEnable, motor_speed);
}
```

Теперь мы рассмотрим важные части этого кода. Сначала мы определяем все выводы, которые мы использовали для подключения ESP8266 к драйверу двигателя L293D:

```
int motorOnePlus = 12;
int motorOneMinus = 13;
int motorOneEnable = 14;

int motorTwoPlus = 5;
int motorTwoMinus = 16;
int motorTwoEnable = 4;
```

В функции скетча setup () мы устанавливаем все эти выводы как выходы:

```
pinMode(motorOnePlus, OUTPUT);
pinMode(motorOneMinus, OUTPUT);
pinMode(motorOneEnable, OUTPUT);

pinMode(motorTwoPlus, OUTPUT);
pinMode(motorTwoMinus, OUTPUT);
pinMode(motorTwoEnable, OUTPUT);
```

Внутри функции loop () мы сначала устанавливаем оба двигателя, чтобы они двигались вперед со скоростью, равной половине их максимальной скорости:

```
setMotorOne(true, 500);
setMotorTwo(true, 500);
```

Позже мы также останавливаем двигатели, прежде чем запускать их снова. Давайте теперь посмотрим на функцию, используемую для установки первого двигателя:

```
void setMotorOne(boolean forward, int motor_speed){
     digitalWrite(motorOnePlus, forward);
  digitalWrite(motorOneMinus, !forward);analogWrite
  (motorOneEnable, motor_speed);
}
```

Как видите, эта функция состоит из двух частей: одна для направления, а другая для скорости. Направление устанавливается путем подачи двух противоположных логических сигналов на контакты + и - L293D для этого конкретного двигателя. Что касается скорости, мы просто подаем сигнал ШИМ на соответствующий вывод с помощью функции analogWrite ().

Обратите внимание, что с помощью этой функции скорость может быть установлена от 0 до 1023.
Функция управления другим двигателем действительно похожа:

```
void setMotorTwo(boolean forward, int motor_speed){
   digitalWrite(motorTwoPlus, forward);
   digitalWrite(motorTwoMinus, !forward);
   analogWrite(motorTwoEnable, motor_speed);
}
```

Пришло время протестировать скетч и заставить моторы двигаться! Прежде чем что-либо делать, убедитесь, что робот стоит на небольшой платформе, чтобы колеса не касались земли. Иначе вы можете получить неприятный сюрприз, когда колеса начнут крутиться! Также проверьте, что аккумулятор подключен к роботу.

Затем загрузите код в робота. Вы должны увидеть, что колеса быстро начнут вращаться в том же направлении, прежде чем останавливаться, а затем снова начинать виток .

Если колеса вращаются в разных направлениях, убедитесь, что все соединения выполнены правильно. Это важно, поскольку мы хотим, чтобы робот двигался вперед, когда мы применяем одну и ту же команду к обоим двигателям.

Подключение робота к облаку

Теперь, когда мы уверены, что колеса работают правильно, мы собираемся подключить робота к облачной платформе aREST, чтобы им можно было управлять из любой точки мира.
Поскольку скетч довольно большой и многое берет из эскиза моторного теста, который мы видели ранее, я остановлюсь здесь только на самых важных моментах. Конечно, вы можете найти полный набросок книги в репозитории GitHub.

Он начинается с импорта соответствущих библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

Затем мы создаем клиента для связи с облачным сервером aREST:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Также создаем экземпляр клиента aREST:

```
aREST rest = aREST(client);
```

Затем, как мы видели в предыдущих главах, вам нужно присвоить устройству уникальный идентификатор ID:

```
char* device_id = "40ep12";
```

Здесь также необходимо указать имя и пароль Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-password";
```

Затем нам нужно определить набор функций для удаленного управления роботом. Здесь я буду использовать одну функцию для каждой основной команды робота: стоп, вперед, назад, вправо и влево.

Сначала нам нужно объявить все эти функции:

```
int stop(String command);
int forward(String command);
int left(String command);
int right(String command);
int backward(String command);
```

В функции setup () мы присваиваем плате идентификатор, а также даем ей имя:

```
rest.set_id(device_id);
rest.set_name("robot");
```

Нам нужно предоставить функции библиотеке aREST, чтобы их можно было вызывать удаленно:

```
rest.function("forward", forward);
rest.function("stop", stop);
rest.function("right", right);
rest.function("left", left);
rest.function("backward", backward);
```

После этого подключаем плату к сети Wi-Fi:

```
WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
```

В функции loop () мы подключаемся к облаку следующим образом:

```
    rest.handle(client);
```

Теперь давайте посмотрим на одну из тех функций, которые мы используем, чтобы заставить робота двигаться.
Это детали функции, заставляющей робота двигаться вперед:

```
   int forward(String command) {

     setMotorOne(true, 1000);
     setMotorTwo(true, 1000);

   }
```

Если вы вспомните функции из предыдущего раздела, это просто установка обоих двигателей на движение в одном направлении на (почти) полной скорости.

Пришло время протестировать робота! Убедитесь, что вы взяли весь код из папки с кодами, присвоили проекту уникальный идентификатор, а также изменили учетные данные Wi-Fi внутри кода. Также убедитесь, что аккумулятор подключен к роботу.

Затем загрузите код на плату. На этот раз после загрузки ничего не должно произойти. Теперь, чтобы робот стал автономным, отключите его от кабеля FTDI и подключите вывод V_{bat} к источнику питания батареи (чип Adafruit ESP8266 может без проблем обрабатывать 7 В). Если у вас есть еще один ESP8266, вам может потребоваться его питание от внешнего источника питания.

Теперь перейдите в свой любимый веб-браузер и введите:

```
https://cloud.arest.io/40ep12/id
```

Конечно, вам нужно заменить идентификатор ID на тот, который вы указали в коде. Вы должны получить следующий ответ:

```
{
  "id": "40ep12",
  "name": "robot",
  "connected": true
}
```

Затем убедитесь, что вокруг робота есть свободное место, и введите:

```
https://cloud.arest.io/40ep12/forward
```

Вы должны получить в своем браузере подтверждение того, что функция была выполнена, и вы также должны увидеть, как робот движется вперед!
Чтобы остановить это, просто введите:

```
https://cloud.arest.io/40ep12/stop
```

Это должно немедленно остановить робота. Теперь вы также можете поиграть с другими функциями, чтобы заставить робота повернуть направо или налево или вернуться назад. Поздравляем, теперь вы можете управлять своим роботом из любой точки планеты!

Управление роботом с приборной панели

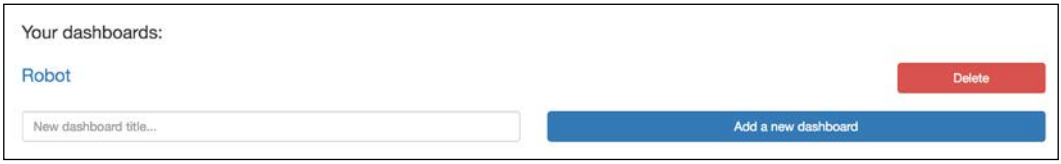
То, что мы уже достигли, прекрасно, но еще не идеально. Управлять роботом из веб-браузера не так удобно, особенно для выполнения быстрых действий, например, для поворота робота на определенный угол.

Чтобы легко управлять роботом, мы снова будем использовать функции панели инструментов aREST, чтобы мы могли управлять роботом с помощью кнопок.

Если это еще не сделано, создайте аккаунт по адресу:

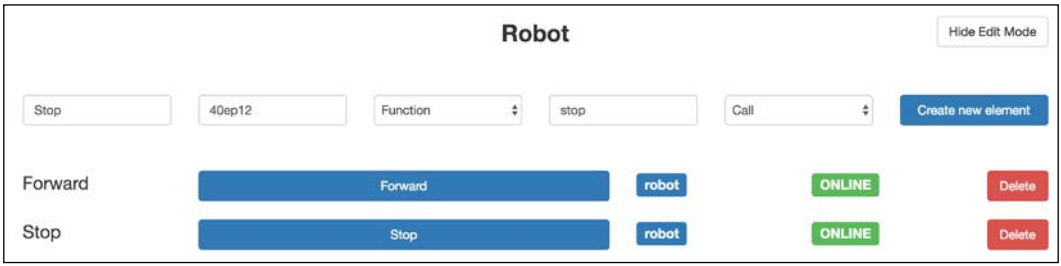
```
http://dashboard.arest.io/
```

Оттуда создайте новую панель для вашего робота:

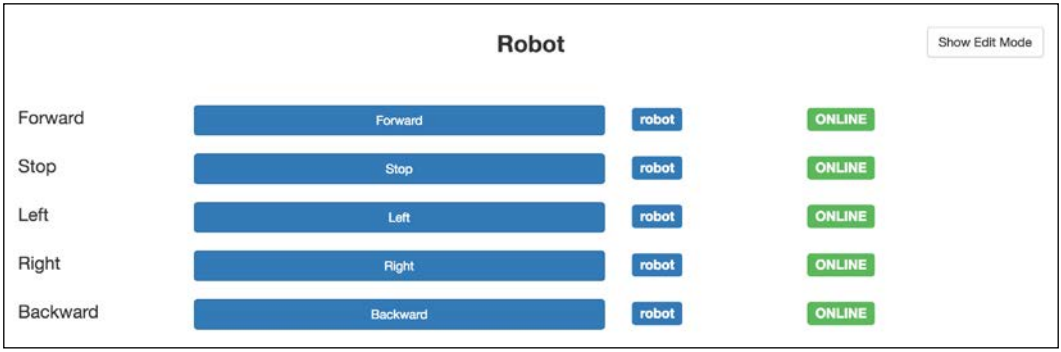


Теперь внутри только что созданной панели инструментов мы собираемся создать кнопку для первой функции: движение вперед. Для этого создайте новый элемент с функцией в качестве типа и forward - вперед в качестве функции для вызова:

Затем сделайте то же самое для функции остановки:



После этого проделаем ту же операцию со всеми остальными функциями:



Теперь вы можете сразу же попробовать кнопки; всякий раз, когда вы нажимаете кнопку, он должен немедленно выполнять правильное действие на ESP8266.

Резюме

В этой главе мы использовали ESP8266 для совершенно другого проекта Интернета вещей, чем в других главах: мы построили мобильного робота. Робот был напрямую подключен к облаку, и мы использовали онлайн-панель для управления им с помощью графического интерфейса. Приятно то, что с его помощью можно управлять роботом из любого места.

Хороший способ улучшить этот проект - добавить к нему датчики, а также управлять этими датчиками из облака. Вы можете, например, добавить ультразвуковые датчики, чтобы «видеть» то, что находится перед роботом.

В следующей главе мы рассмотрим более сложную тему: как развернуть собственный облачный сервер, чтобы вы могли полностью контролировать свой проект Интернета вещей с помощью ESP8266.

13

Создание собственной облачной платформы для управления устройствами ESP8266

Во всех предыдущих главах этой книги мы всегда использовали внешние сервисы для подключения наших проектов ESP8266 к облаку. Например, мы использовали такие сервисы, как IFTTT, Adafruit IO и aREST, для управления и мониторинга наших проектов из облака.

Однако все эти службы управляются другими людьми или компаниями. Это может создать проблемы с безопасностью для некоторых проектов, а также эти службы могут быть отключены в любое время или, по крайней мере, не управляться должным образом. Если вы действительно хотите иметь полный контроль над проектами, подключенными к облаку, единственный выход - развернуть собственный сервер в облаке.

Именно этим мы и займемся в этой главе. Сначала мы собираемся создать облачный сервер, чтобы мы могли развертывать веб-приложения, к которым можно получить доступ из любого места.

Затем мы узнаем, как развернуть собственный облачный сервер aREST на только что созданном сервере. Наконец, в качестве теста мы подключим простой проект ESP8266 к вашему собственному серверу. Давайте начнем!

Аппаратные и программные требования

Давайте сначала посмотрим, что нам нужно для этого проекта. Поскольку мы просто хотим протестировать соединение между платой ESP8266 и вашим собственным облачным сервером, мы будем упрощать задачу.

Для ESP8266 я снова решил использовать модуль Adafruit ESP8266 вместе с USB-модулем FTDI.

Мы просто подключим к этой плате два типа компонентов: светодиод и датчик DHT11.

Конечно, вам понадобятся обычные макетные платы и перемычки.

Это список всех компонентов, которые будут использоваться в этой главе:

- Модуль Adafruit ES8266
- USB-модуль FTDI
- LED
- Резистор 330 Ом
- Датчик DHT11
- Макетная плата
- Перемычки

Что касается программного обеспечения, вам потребуются IDE Arduino и библиотека aREST, которые мы уже использовали ранее в этой книге. Вам также понадобятся библиотеки PubSub и Adafruit DHT.

Аппаратная конфигурация

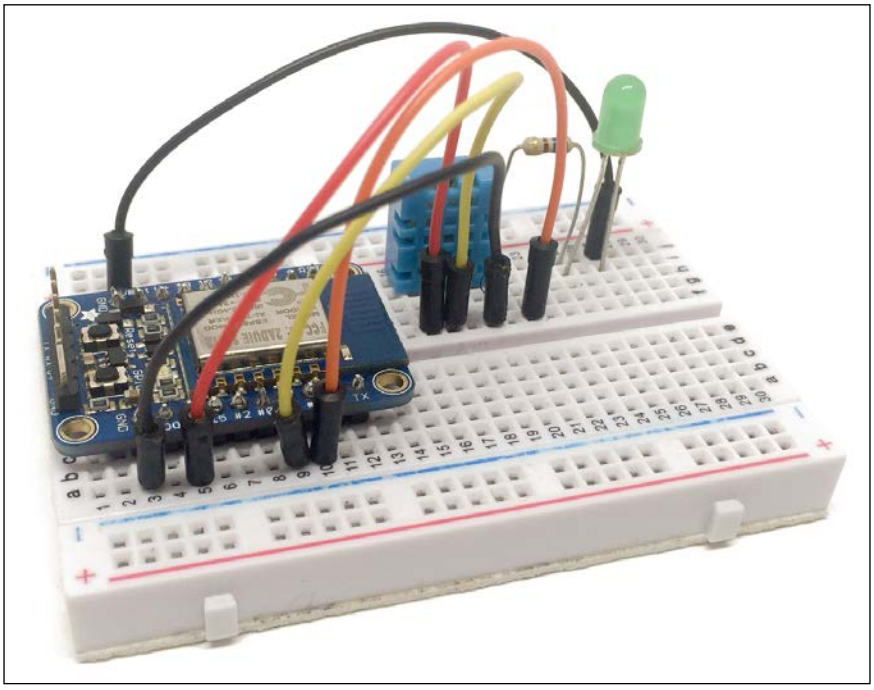
Теперь собираемся собрать проект. Аппаратное обеспечение здесь действительно простое, поскольку мы хотим только протестировать соединение между проектом и нашим собственным облачным сервером, который мы развернем позже в этой главе.

Просто поместите плату ESP8266 на макетную плату, а затем подключите к ней коммутационную плату FTDI.

Для светодиода просто подключите его последовательно с резистором, так чтобы самый длинный вывод светодиода был подключен к резистору. Затем подключите оставшийся контакт резистора к контакту 5 платы ESP8266, а оставшийся контакт светодиода - к контакту GND.

Как только это будет сделано, просто поместите датчик DHT11 на макетную плату. Затем подключите левый контакт к VCC, правый контакт к, а контакт рядом с VCC к контакту 4 на микросхеме ESP8266.

Это конечный результат:



Создание облачного сервера

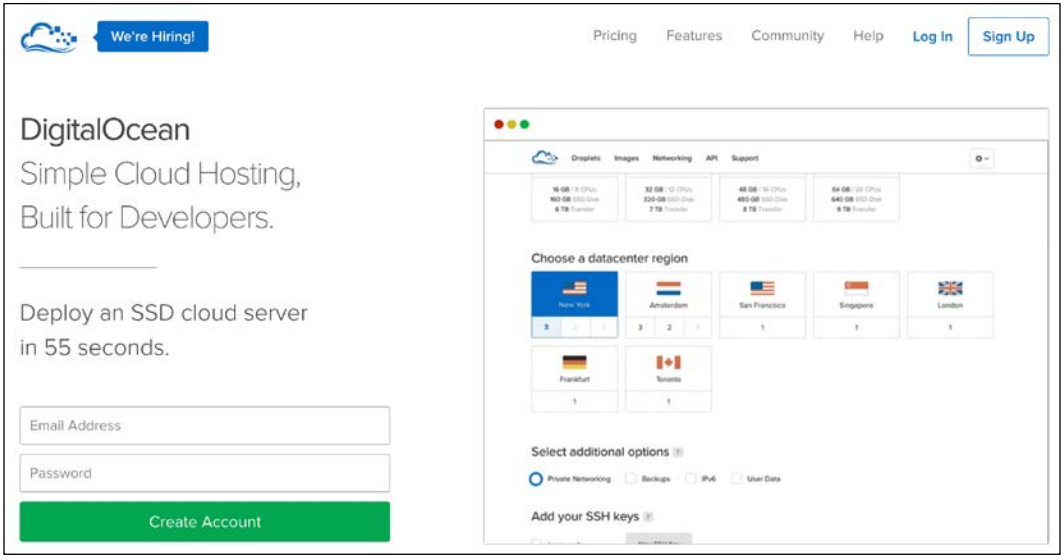
Теперь мы собираемся сделать первый шаг к подключению платы к вашему собственному облачному серверу: создать сам сервер.

Вы можете запустить программное обеспечение, которое мы увидим позже, на вашем собственном компьютере, но тогда вы не сможете получить доступ к своим проектам ESP8266 удаленно. Вот почему вам необходимо развернуть собственный сервер с поставщиком облачного сервера. Если у вас уже есть такой сервер, на котором можно запустить приложение Meteor (фреймворк, который мы собираемся использовать), вы можете просто пропустить этот раздел.

Есть много поставщиков серверов, но я рекомендую Digital Ocean. Он быстрый, дешевый и имеет очень простой в использовании интерфейс. Вы можете найти его по адресу:

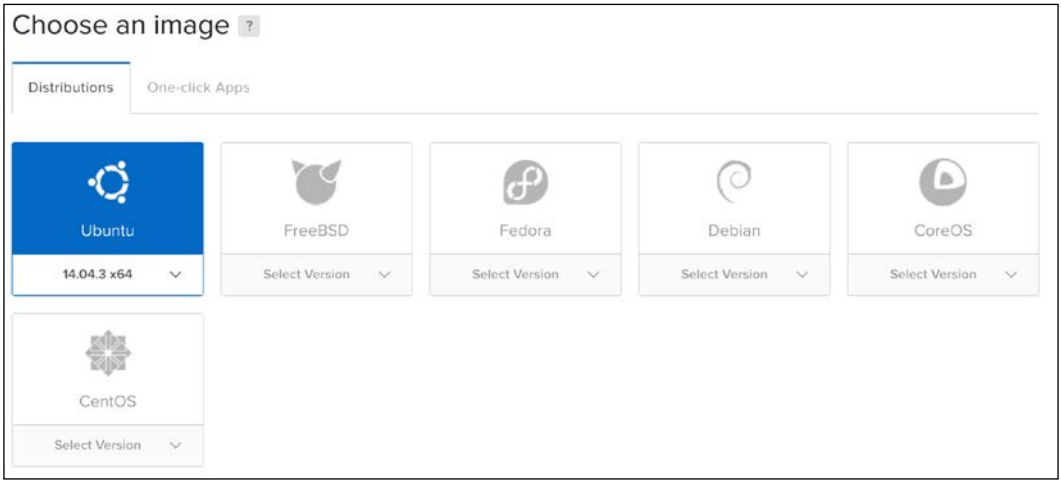
`https://www.digitalocean.com/`

Зайдя на их веб-сайт, создайте новый аккаунт. Затем создайте новый Droplet - имя сервера в Digital Ocean. Вам будет предложено выбрать, где вы хотите развернуть дроплет (выберите то, что ближе всего к устройствам, которые вы хотите использовать):

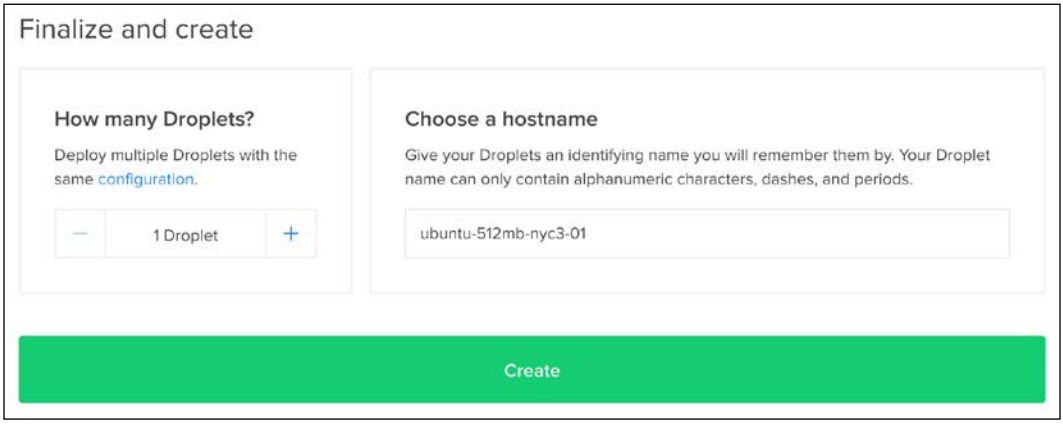


Затем вам нужно выбрать ежемесячный план, который также будет определять вычислительную мощность вашего сервера. Поскольку у нас здесь только легкое программное обеспечение, самого дешевого плана более чем достаточно для нашего сервера:

Далее вам нужно выбрать операционную систему для вашего сервера. Просто выберите последнюю стабильную версию операционной системы Ubuntu Linux:



Наконец, завершите создание сервера, присвоив ему имя:



Теперь вы должны увидеть, что ваш сервер создан, и он должен появиться в списке:

Оттуда вам понадобится IP-адрес вашего сервера, который мы собираемся использовать позже.
Вам также необходимо будет установить root-доступ на вашем сервере. Поскольку это сложный процесс, я рекомендую следовать всем инструкциям здесь:

`https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server`

Код облачного сервера aREST

Пришло время развернуть приложение на только что созданном сервере.

В качестве программы для вашего облачного сервера мы используем облачный сервер aREST, который мы уже использовали ранее.

Он полностью бесплатный и с открытым исходным кодом, и вы можете получить последнюю версию программного обеспечения отсюда:

`https://github.com/marcoschwartz/meteor-aREST-mqtt`

А пока просто скопируйте все файлы из этого репозитория и поместите их в любую папку на вашем компьютере.

Это приложение фактически использует фреймворк Meteor, который представляет собой законченный фреймворк на основе JavaScript и Node.js и может использоваться для создания мощных веб-приложений. Если это еще не сделано, установите Meteor на свой компьютер, перейдя по ссылке:

`https://www.meteor.com/install`

Теперь давайте кратко рассмотрим код облачного сервера aREST. Код довольно сложный, но я просто хотел выделить здесь некоторые части, чтобы вы поняли, как он работает.

Каждый раз, когда устройство подключается к серверу, программа сначала проверяет, знает ли сервер устройство. В этом случае мы настраиваем устройство на подключение, чтобы оно было доступно из облака. Если устройство неизвестно, оно добавляется в базу данных на сервере.

Это фрагмент кода, который обрабатывает эти функции:

```
Server.on('clientConnected', Meteor.bindEnvironment(function(client) {

  console.log('client connected', client.id);
```

```
  // Уже существует?
  if (Devices.find({clientId: client.id}).fetch().length == 0) {

    // Вставить в БД
    console.log('New device detected');
    device = {
      clientId: client.id,
      connected: true
    }
    Devices.insert(device);
  }
  else {
    console.log('Existing device detected');

    // Обновить статус устройства
    Devices.update({clientId: client.id}, {$set: {"connected":
true}});

  }

}) );
```

Всякий раз, когда пользователь хочет получить доступ к устройству, пользователь вводит команду в веб-браузере. Затем облачный сервер сначала сопоставит эту команду с доступными командами, например:

```
Router.route('/:device', {
```

Затем сервер проверяет, существует ли устройство:

```
var currentDevice = Devices.findOne({clientId: device});
```

Если устройство существует и подключено в данный момент, сервер готовит сообщение для отправки на устройство, например:

```
var message = {
  topic: currentDevice.clientId + '_in',
  payload: '',
  qos: 0,
  retain: false
};
```

Затем, как только ответ приходит с устройства, мы отправляем его пользователю в формате JSON:

```
answer = sendMessage(message);
this.response.setHeader('Content-Type', 'application/json');
this.response.end(answer);
```

Развертывание сервера

Теперь мы собираемся развернуть программу на вашем облачном сервере. Во-первых, вам нужно установить Node.js, если это еще не сделано. Для этого просто следуйте инструкциям по адресу: `https://nodejs.org/en/`

Затем мы можем установить программу под названием Meteor Up, которое действительно упростит процесс развертывания приложения на нашем веб-сервере.

Зайдите в терминал и введите:

```
sudo npm install -g mup
```

Затем перейдите в папку, в которую вы поместили все файлы приложения, и инициализируйте Meteor Up с помощью:

```
mup init
```

Это создаст файл с именем mup.json внутри папки, в которой вы сейчас находитесь. Вот как этот файл будет выглядеть:

```
{
  // Информация об аутентификации сервера
  "servers": [
    {
      "host": "0.0.0.0",
      "username": "root",
      //"password": "password"
      // или файл pem (аутентификация на основе ssh)
      "pem": "~/.ssh/id_rsa"
    }
  ],

  // Установить MongoDB на сервер, не разрушает локальную настройку
  // MongoDB в будущем.
  "setupMongo": true,
```

```
// ВНИМАНИЕ: требуется Node.js! Пропускате, только если у вас уже есть
    Node.  js установлен на сервере.
  "setupNode": true,

  // ВНИМАНИЕ: Если параметр nodeVersion опущен, по умолчанию будет
установлено значение 0.10.36. Не используйте v, только номер версии.
  "nodeVersion": "0.10.41",

  // Установите PhantomJS на сервер
  "setupPhantom": true,

  // Показывать индикатор выполнения во время загрузки пакета на
сервер.
  // В некоторых редких случаях может вызвать ошибку, если
задано значение true, например, в Shippable CI.
  "enableUploadProgressBar": true,

  // Название приложения (без пробелов)
  "appName": "arest",

  // Расположение приложения (локальный каталог)
  "app": ".",

  // Настроить среду
  "env": {
    "ROOT_URL": "http://localhost",
    "PORT": 3000
  },

  // Meteor Up проверяет, выходит ли приложение в сеть сразу
после развертывания.
  // before mup checks that, it will wait for no. of seconds
configured below
  "deployCheckWaitTime": 120
}
```

В этом файле вам нужно будет изменить две вещи. Первый - установить IP-адрес вашего сервера Digital Ocean:

```
"servers": [
    {
      "host": "0.0.0.0",
      "username": "root",
```

```
        //"password": "password"
        // или файл pem (аутентификация на основе ssh)
        "pem": "~/.ssh/id_rsa"
      }
  ]
```

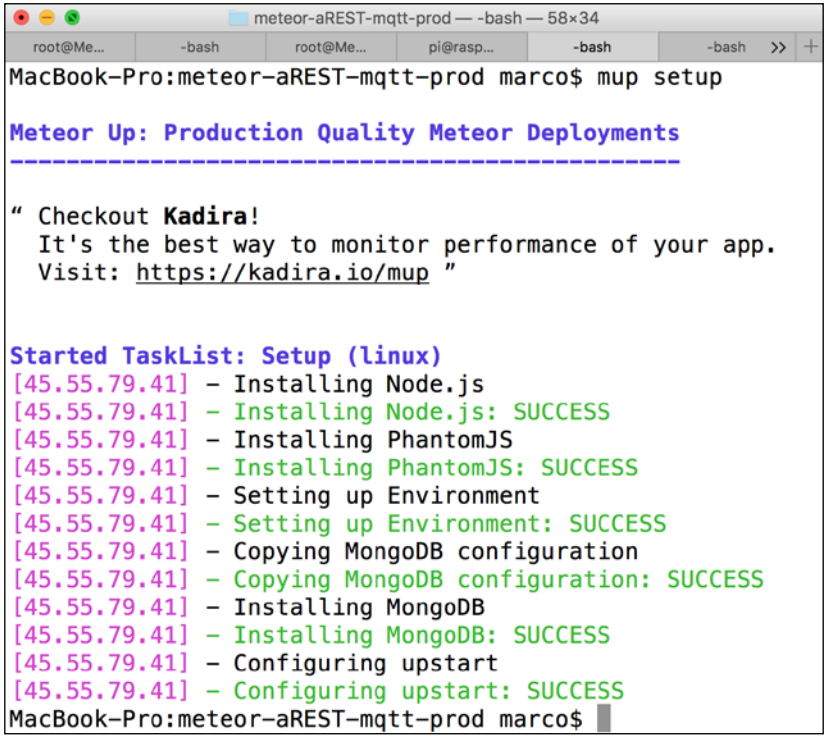
Вы также можете изменить порт, чтобы приложение запускалось на вашем сервере. По умолчанию это будет 3000, но вы можете изменить его, если у вас есть несколько приложений, работающих на одном сервере:

```
"env": {
    "ROOT_URL": "http://localhost",
    "PORT": 3000
  }
```

Как только это будет сделано, сохраните файл и введите следующую команду:

```
mup setup
```

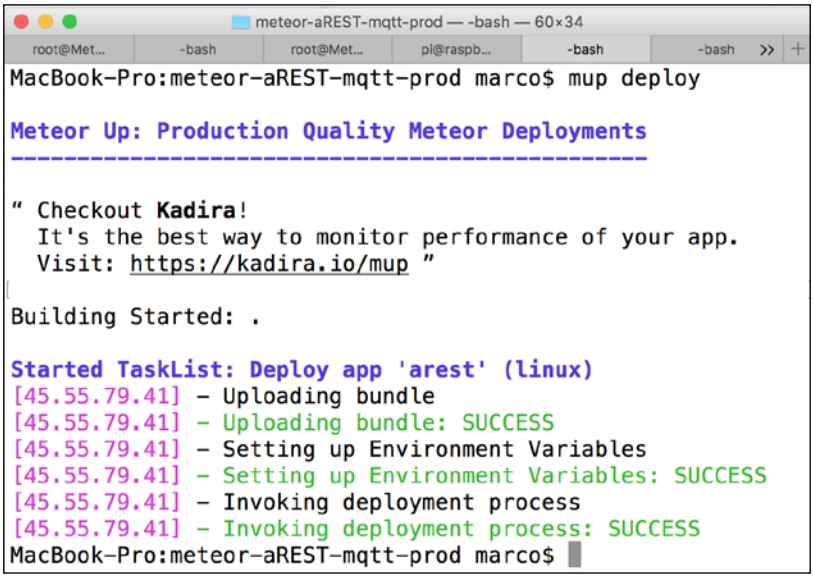
Это инициализирует ваш сервер, чтобы можно было развернуть приложение Meteor:



Как только вы увидите последнее сообщение SUCCESS - УСПЕХ, вы можете продолжить и развернуть приложение с помощью:

```
mup deploy
```

Это запустит процесс развертывания:



Как только вы увидите последнее сообщение SUCCESS - УСПЕХ, выделенное зеленым цветом, поздравляем, теперь у вас есть собственный облачный сервер, развернутый и готовый к использованию для управления вашими IoT-устройствами!

Подключение платы ESP8266 к вашему облачному серверу

Теперь мы действительно хотим протестировать сервер, который мы только что развернули в облаке. Для этого мы осуществим аппаратные настройки,чтобы подключиться к вашему собственному облачному серверу:

1. Во-первых, нам нужно включить все необходимые библиотеки:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
#include "DHT.h"
```

2. Затем мы определяем, к какому выводу подключен датчик DHT:

```
#define DHTPIN 4
#define DHTTYPE DHT11
```

3. Далее мы создаем экземпляр датчика DHT:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

4. Создаем клиентов для подключения к вашему облачному серверу:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

5. Затем мы создаем экземпляр aREST. Здесь вам нужно передать IP-адрес вашего удаленного облачного сервера в качестве аргумента:

```
aREST rest = aREST(client, "192.168.0.103");
```

6. Вам также необходимо указать ID вашего текущего устройства:

```
char* device_id = "01e47f";
```

7. Затем установите имя Wi-Fi и пароль скетча:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

8. Мы также создаем две переменные для хранения значений проводимых нами измерений:

```
float temperature;
float humidity;
```

9. В функции скетча setup () мы инициализируем датчик DHT:

```
dht.begin();
```

10. Мы также предоставляем aREST API две измерительные переменные:

```
rest.variable("temperature", &temperature);
rest.variable("humidity", &humidity);
```

11. В функции скетча loop () мы измеряем данные, поступающие от датчика, а также поддерживаем соединение с вашим облачным сервером открытым:

```
// Считывание температуры и влажности
humidity = dht.readHumidity();
// Считать температуру по Цельсию
temperature = dht.readTemperature();

// Подключение к облаку
rest.handle(client);
```

Пришло время протестировать скетч и посмотреть, сможет ли он подключиться к только что развернутому облачному серверу! Чтобы проверить, так ли это, возьмите код из папки с кодами. Затем измените имя и пароль Wi-Fi внутри кода и загрузите его на плату.

Затем откройте Serial monitor, чтобы проверить, можно ли установить соединение с вашим облачным сервером:

Если вы видите «Connected to MQTT server - Подключено к серверу MQTT», это означает, что ваше устройство в настоящее время подключено к серверу, который вы развернули в облаке ранее, и устройство обменивается данными через сервер aREST, используя протокол MQTT. Если нет, вернитесь к процессу, который мы видели в этой главе, и убедитесь, что приложение aREST правильно развернуто на вашем облачном сервере.

Теперь вы можете проверить связь между вашим устройством и облачным сервером, как и раньше с сервером aREST.io. Например, чтобы узнать температуру, введите:

```
http://19.434.34.23/01e47f/temperature
```

Конечно, вам нужно заменить IP-адрес на адрес вашего облачного сервера. Вы должны сразу увидеть ответ в своем браузере:

```
  {
"temperature": 26.00,
"id": "01e47f",
"name": "own_cloud",
"connected": true
  }
```

Вы также можете установить вывод 5 как выход с помощью:

```
http://19.434.34.23/01e47f/mode/5/o
```

Затем включите светодиод:

```
http://19.434.34.23/01e47f/digital/5/1
```

Поздравляем, теперь вы можете подключать устройства к собственному облачному серверу!

Резюме

В этой главе мы увидели, как развернуть собственный сервер в облаке, чтобы вы могли полностью контролировать свои проекты Интернета вещей с помощью чипа ESP8266 Wi-Fi. Мы увидели, как создать свой собственный сервер с помощью Digital Ocean, как развернуть на нем программное обеспечение aREST и, наконец, как подключить один ESP8266 к этому недавно развернутому серверу.

Конечно, теперь вы можете подключить к этому серверу любой из ваших проектов ESP8266. Вы можете просто использовать проекты, которые вы видели в этой книге, которые используют aREST, и подключить их к вашему собственному серверу. Вы также можете повозиться с кодом и создать новые функции для своего серверного приложения. Если это так, не стесняйтесь делиться своими новыми функциями с сообществом!