[10.04.2018]

<<IoT con ESP8266 e NodeRed – Parte 1>>

C&W

</Agenda>

1. **Intro**
2. **ESP8266**
1. Caratteristiche
2. Setup via Arduino IDE
3. LAB: setup Arduino x ESP8266
4. LAB: lettura sensore
5. Advanced1: Deep Sleep
6. Advanced2: SDK Functions
3. **NodeRed**
1. Caratteristiche
2. MQTT
3. LAB: invio dati sensore ad una dashboard

MQTT.ORG

Node-RED

Telegram

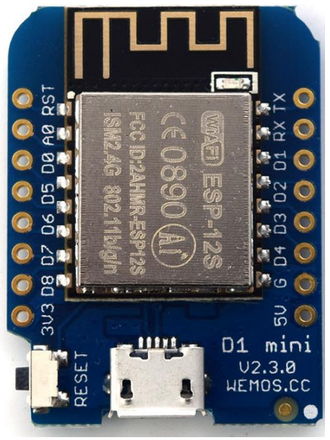
- E’ un SOC (System on a Chip)
- MCU 32bit, Tensilica’s L106 Diamond series (arduino: 8bit ATmega328P)
- 80 o 160Mhz (arduino: 16MHz)
- 36kB SRAM (arduino: 2-8kB)
- Flash da 1 a 16MB, tipicamente 4MB (arduino: 32-256kB)
- GPIO: molto variabile da vendor a vendor, WEMOS ha 11 Dgt I/O assegnabili.
- Analog In: 1 (arduino: 6 analog IN)
- Tensione operativa IO: 3,3v (arduino 5v!)
- DeepSleep

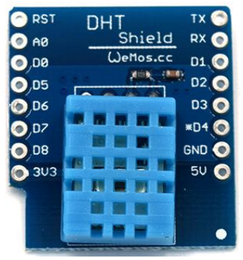
Caratteristiche Tecniche

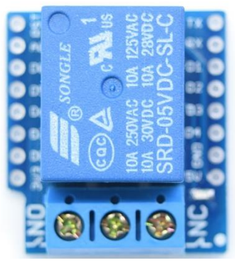
Core	ESP-8266EX
Operating Voltage	3.3V
Digital I/O Pins	11
Analog Input Pins	1(Max input: 3.2V)
Clock Speed	80MHz/160MHz
Flash	4M bytes
Length	34.2mm
Width	25.6mm
Weight	10g

D1 mini Shields:

1. Battery Shield
2. Matrix LED Shield
3. Buzzer Shield
4. Dual Base
5. SHT30 Shield
6. WS2812B RGB Shield
7. ProtoBoard Shield
8. DHT Shield
9. 1-Button Shield
10. Micro SD Card Shield
11. Relay Shield
12. DC Power Shield
13. Tripler Base
14. Motor Shield
15. OLED Shield





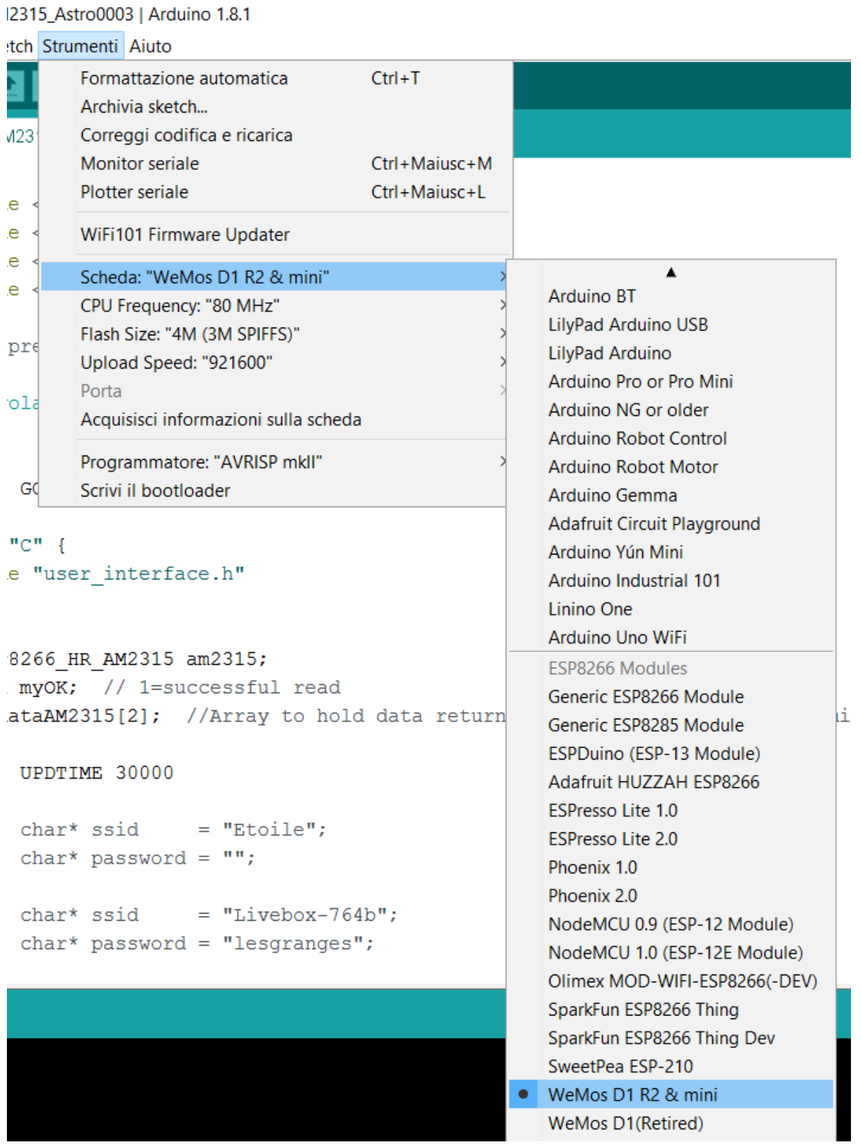


1. Download IDE: https://www.arduino.cc/en/Main/Software

2. Installare IDE

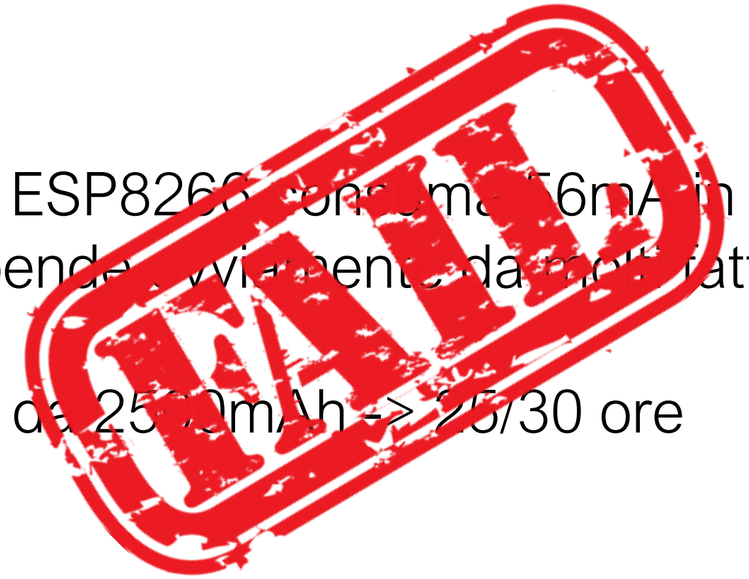
3. File → Impostazioni → URL aggiuntive per gestore schede. Inserire: http://arduino.esp8266.com/stable/package_esp8266com_index.json

4. Strumenti → Schede → gestore schede
5. Selezionare «ESP8266 by ESP8266 Community»
6. ok
7. Selezionare la scheda Wemos D1 mini nel gestore schede.



LAB: setup Arduino x ESP8266

LAB: lettura sensore



Normalmente un ESP8266 consuma 76mA in ricezione e 150/170mA in trasmissione (dipende ovviamente da molti fattori ambientali)

→

Con una batteria da 2500mAh -> 25/30 ore

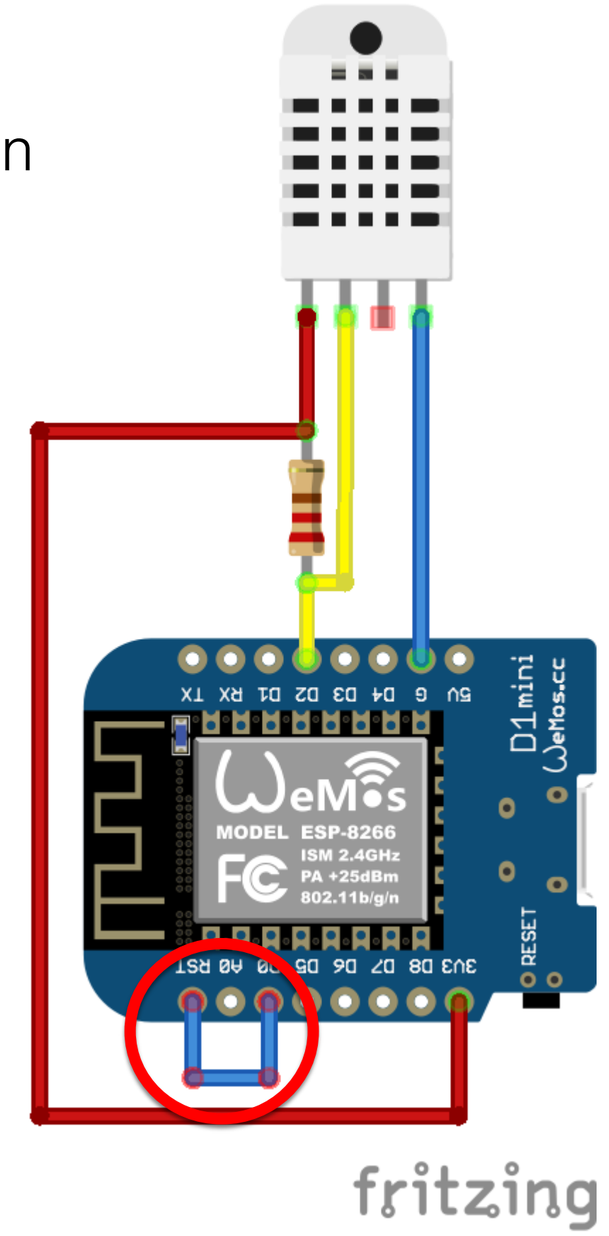
Deep Sleep consuma 10uA (0.010mA)

→

Con una batteria da 2500mAh -> 2/4 anni
(Attenzione con MQTT)

D0 viene alzato dopo N microsecondi con la funzione:
ESP.deepSleep(sleepSeconds * 1000000);

Basta quindi connettere D0 a RST!

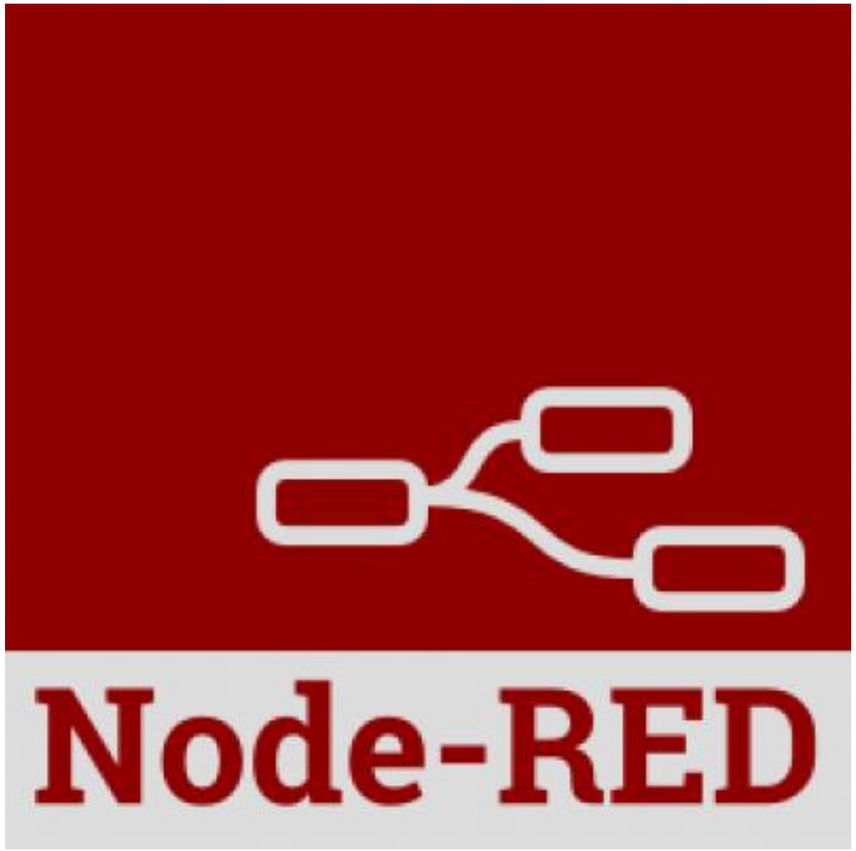


Parameter	Typical	Unit
Tx 802.11b, CCK 11Mbps, P_{OUT}=+17dBm	170	mA
Tx 802.11g, OFDM 54Mbps, P_{OUT} =+15dBm	140	mA
Tx 802.11n, MCS7, P_{OUT} =+13dBm	120	mA
Rx 802.11b, 1024 bytes packet length , -80dBm	50	mA
Rx 802.11g, 1024 bytes packet length, -70dBm	56	mA
Rx 802.11n, 1024 bytes packet length, -65dBm	56	mA
Modem-Sleep	15	mA
Light-Sleep	0.5	mA
Power save mode DTIM 1	1.2	mA
Power save mode DTIM 3	0.9	mA
Deep-Sleep	10	uA
Power OFF	0.5	uA

https://www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf

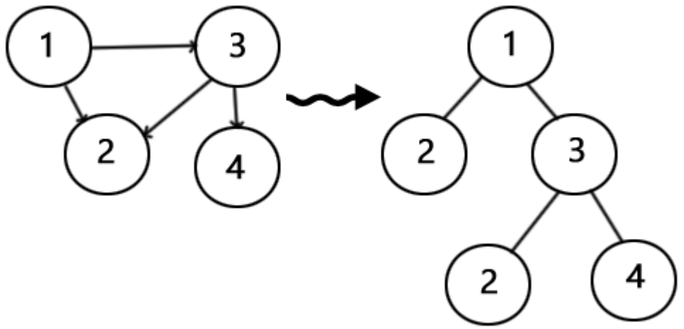
Esempi:

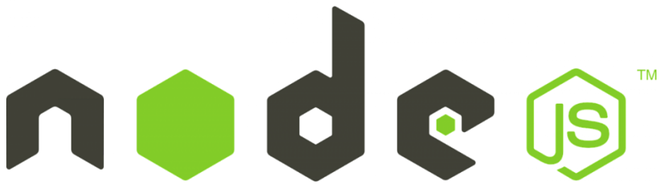
1. wifi_station_scan(…)
2. system_get_cpu_freq();
3. sntp_get_current_timestamp();
4. os_random();
5. system_get_vdd33();

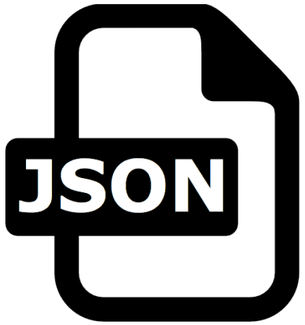

Node-RED

Caratteristiche

- Paradigma «dataflow» (functional) e «event driven»
- Altri esempi: Simulink, VHDL, LabVIEW…
- DAG: Directed Acyclic Graph (limite: ∀ nodo ∃! Ingresso)
- Runtime costruito su NodeJS (distribuito anche via Docker)
- Blocchi possono essere aggregati in superblocchi
- Descritto in JSON → facilita condivisione → community attiva (GitHub/Slack/SOF)
- Community molto attiva (https://flows.nodered.org/)



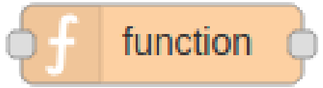




Tipologie di nodi

Input

Output

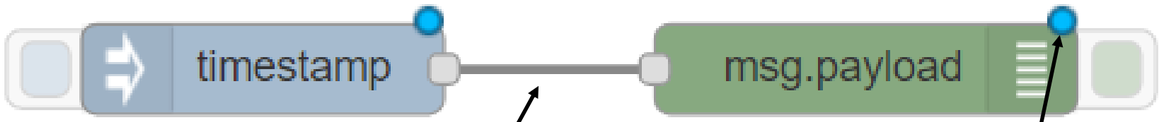


Processing

Payload

- I nodi si passano oggetti JSON «msg»
- Il payload del messaggio è quindi «`msg.payload`»
- Esempio: un nodo che restituisce il clima corrente restituirà la temperatura in celsius «`tempc`». Per leggere la temperatura corrente quindi: `var temperatura = msg.payload.tempc;`

Primo Flow



Il nodo «inject» produce al click un msg il cui payload è il current timestamp (epoch)

msg

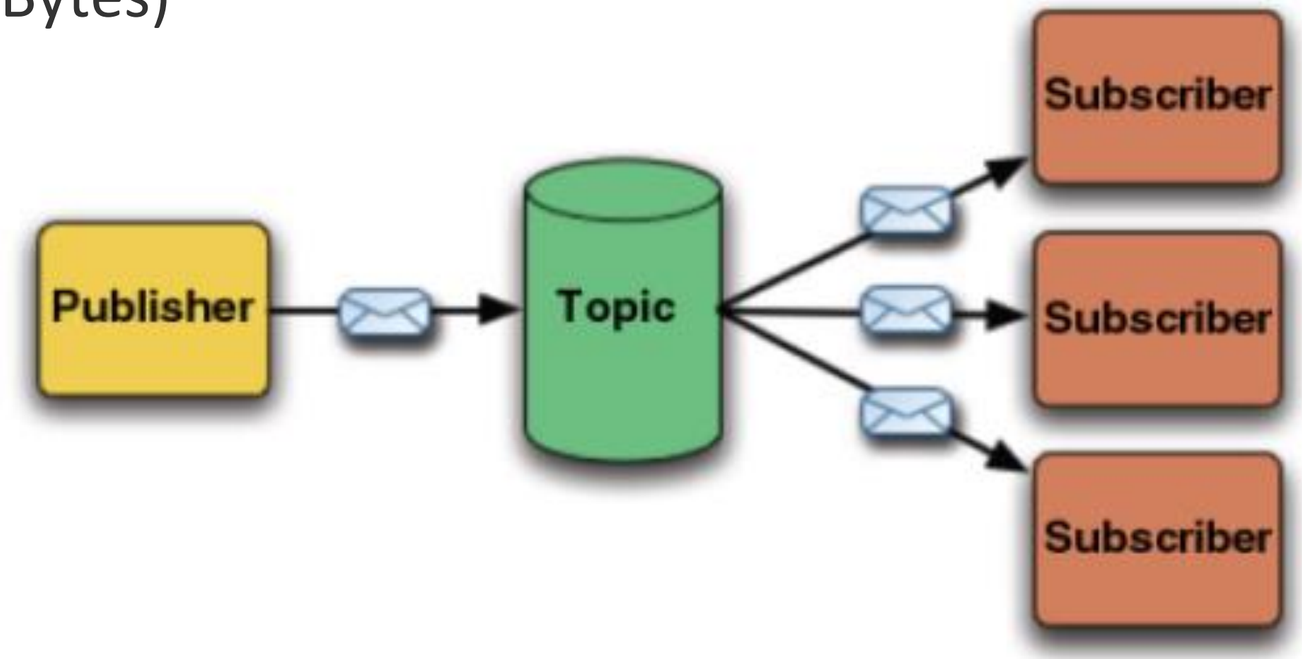
Significa che non è stato ancora effettuato il «deploy»

LAB: esempio nodered


MQTT
.ORG

Caratteristiche

- E’ un protocollo di trasporto
- Over TCP
- Semplice
- Paradigma Pub/Sub (publish subscribe) → Ottimo per IoT
- Canale persistente (connessione attiva anche non in trasmissione) → Nat.T
- Basso overhead protocollare (2Bytes)
- 3 livelli di QoS (0, 1 e 2)
- LWT (last will and testament)



Libreria Arduino

http://pubsubclient.knolleary.net/index.html

Limitazioni:
Pub: QOS 0 – Sub QOS 0,1
Messaggi: **128 bytes** max (compreso header)

Macro Step del nostro nodo IoT

Connessione Wifi	`WiFi.begin(ssid, password);`
Connessione MQTT Broker	`boolean connect (clientID, willTopic, willQoS, willRetain, willMessage)`
Loop	`void loop()`
Check Connessione (riconnessione)	`if (!client.connected()) { reconnect(); }`
Lettura sensori ogni N secondi	`Dipende dal sensore`
Invio dati al broker (publish)	`int publish (topic, payload)`

Esempio codice

+

LAB: Invio dati ad un broker

NR + RPI

Nodered su raspberry? SI!

1. Nodejs: portabile senza (!) problemi! (c'è anche docker image)
2. RPI funziona in un contesto locale: accesso a risorse locali (hue, tradfri, domotica…)
3. RPI può fare da «gateway» verso una istanza NR cloud
4. Attenzione al NAT -> per essere accessibile da esterno aprire porte e mettere in sicurezza

NR + AWS

Nodered su AWS? SI!

1. T2.micro gratis 12mesi (poi 77€/anno…)
2. Accessibile da ovunque
3. Non teme blackout o SD card fail…
4. Prestazioni
5. Possibilità di usare DB evoluti x storicizzazione dati

Kalpa Srl
Via Carducci 39
20099
Sesto San Giovanni (MI)

Tel: +39 02.87187579
email: info@kalpa.it
Web: www.kalpa.it