LoRaWAN IoT Prototyping with Mbed & Semtech LoRa Shield

thomas.amberg@yaler.net

Internet of Things (IoT)

Computers with **sensors** and **actuators**, **connected** through Internet protocols.

Instead of just accessing and editing virtual resources, we can now measure and manipulate **physical properties**.

For developers: IoT = physical objects with an API.

IoT reference model

Prototyping like a hacker

Everybody can do it!

Focus on **results**, not process.

Learn by **doing**, get ideas on the way.

Iterate! Failing is learning. Move on, try again.

Constraints: limits of the design space - embrace them.
Affordance: what an object is capable of vs intended for.

Topics of this workshop

1) **Getting started** with Mbed
2) **Using sensors and actuators** with Mbed
3) **Connecting to LoRaWAN** with ThingPark
4) **Running a Web service** with NodeJS
5) **Storing sensor data** with ThingSpeak
6) **Controlling your device** with Curl
7) **Mash-ups with 3rd party services** on IFTTT

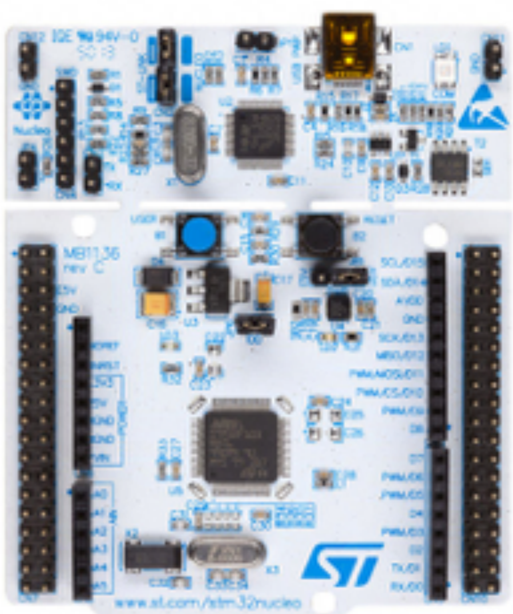
Questions? Just ask / Use Google / Help each other

1) Getting Started

How to set up the Mbed hardware – the basics of embedded programming, step by step.

Hardware

This tutorial is based on the STM32 **Nucleo** L152RE **Mbed board** and the Semtech SX 1276 **LoRa shield**



Note: For the first part we just need the Mbed board

Embedded programming

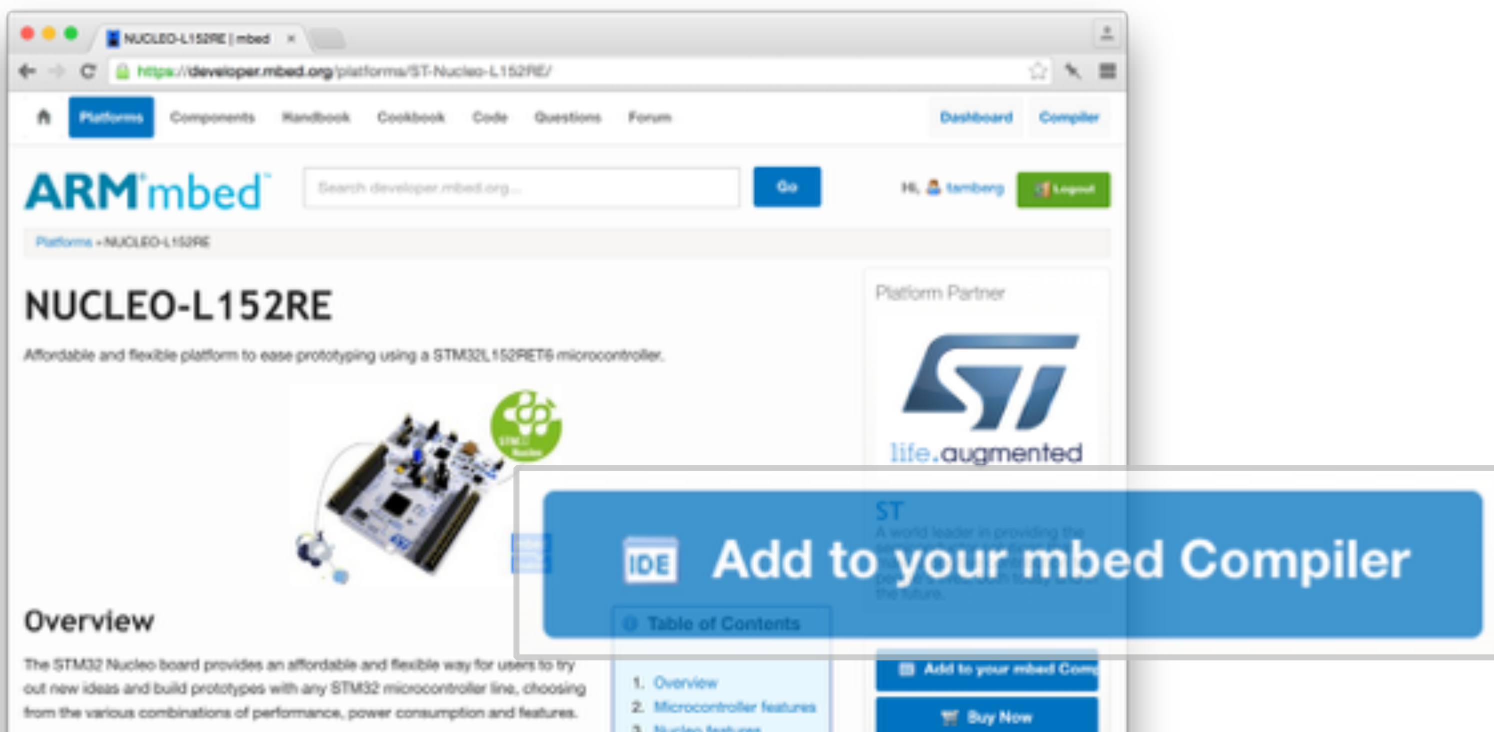
The **IDE** (**I**ntegrated **D**evelopment **E**nvironment) allows you to **program** your board, i.e. "make it do something new".

You **edit** a program on your computer, then **upload** it to your board where it's stored in the program memory (flash) and **executed** in RAM.

Note: Once it has been programmed, your board can run on its own, without another computer.

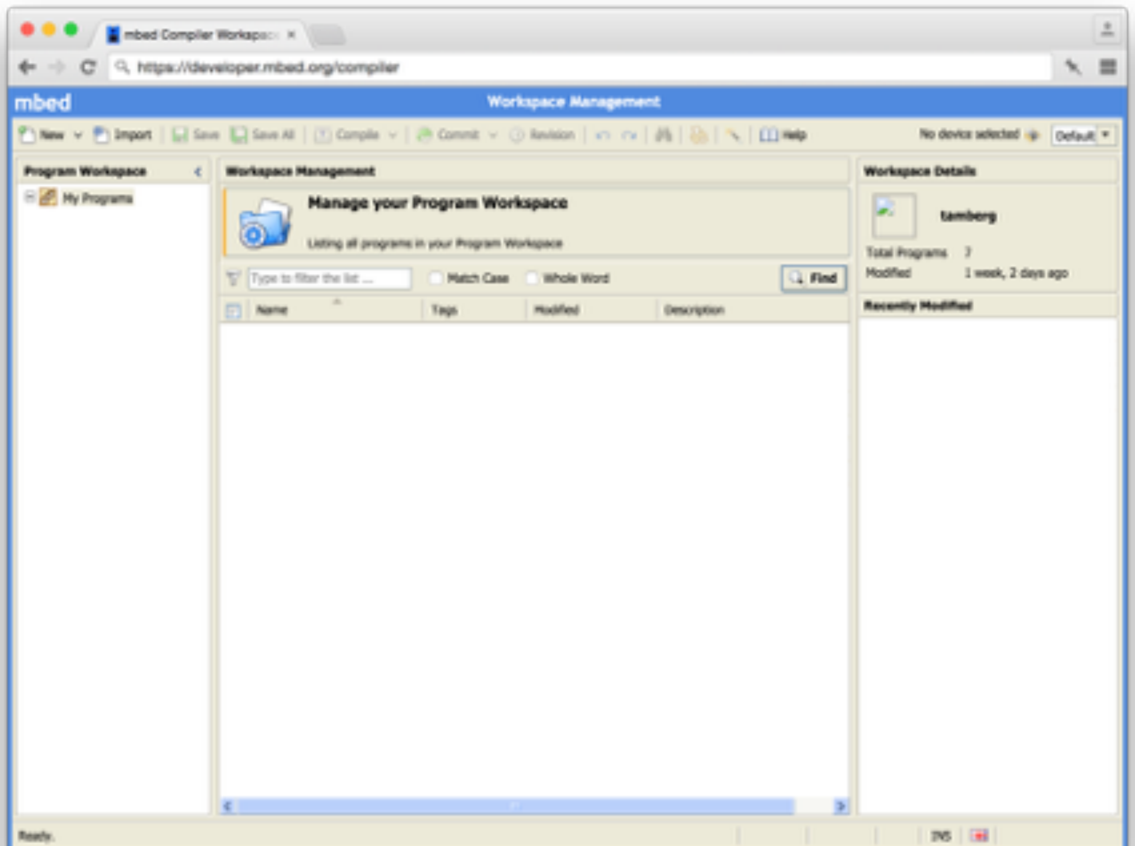
Adding STM32 Nucleo L152RE

Sign up at https://developer.mbed.org/ then open https://developer.mbed.org/platforms/ST-Nucleo-L152RE/ and add it to your compiler.



Opening the Mbed IDE and compiler

https://developer.mbed.org/compiler



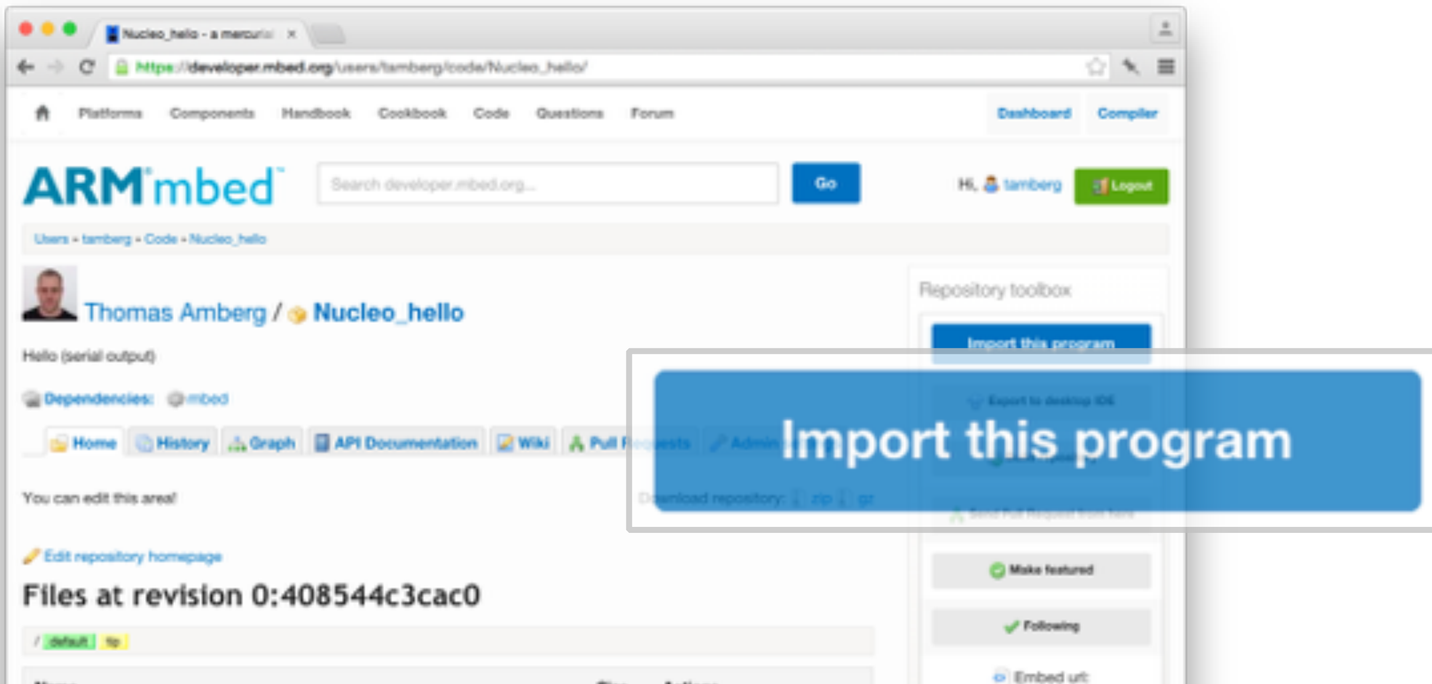
Here you'll edit your programs

Connect your Nucleo via USB, it's also a "disk"

Once compiled a program can be copied to it.

Importing a program to the IDE

https://developer.mbed.org/users/tamberg/code/Nucleo_hello/ (URL depends on the example)



Note: after import, select the **main.cpp** file to open it.

Hello (serial output)

```
#include "mbed.h"
int main() {
    while (1) { // loop
        printf("Hello\r\n");
        wait_ms(1000); // 1 sec
    }
}
```

This program is written in C

\r\n means RETURN in C

while(1) is an endless loop

Note: compile this code and upload it to the device (just drag .bin to the NUCLEO disk), then check the next slide.

Serial output with Nucleo on Mac

Open a **terminal**, connect the board to **USB**, and type

$ ***screen /dev/tty.u***

Then hit *TAB* to find the USB device name

$ *screen /dev/tty.usbmodem1431*

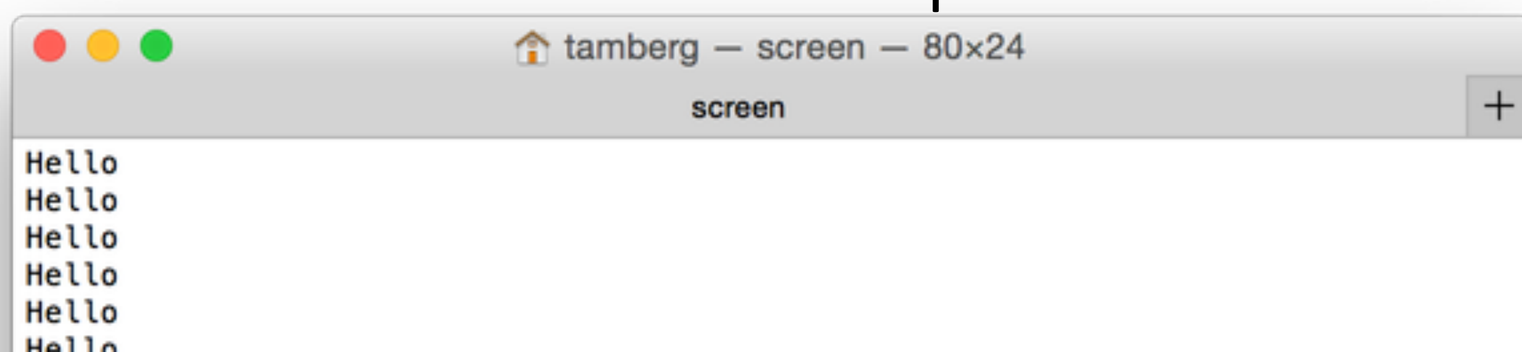
Add the baud rate matching your board

$ *screen /dev/tty.usbmodem1431* ***9600***

And hit RETURN to see the output

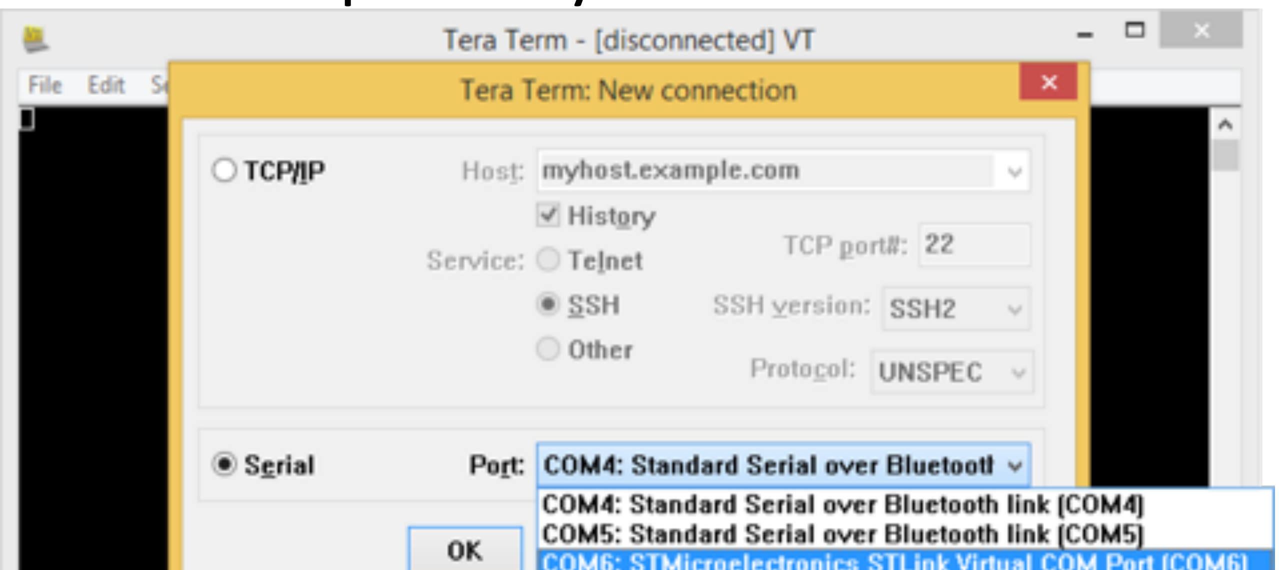
On Linux it's /dev/ttyACM

Windows: next slide.



Serial output with Nucleo on PC

Install the **STLink driver** from https://developer.mbed.org/teams/ST/wiki/ST-Link-Driver then install **TeraTerm** from https://en.osdn.jp/projects/ttssh2/releases/ and select the **serial** COM port of your Mbed device to see output.



No COM port? Try a reboot.

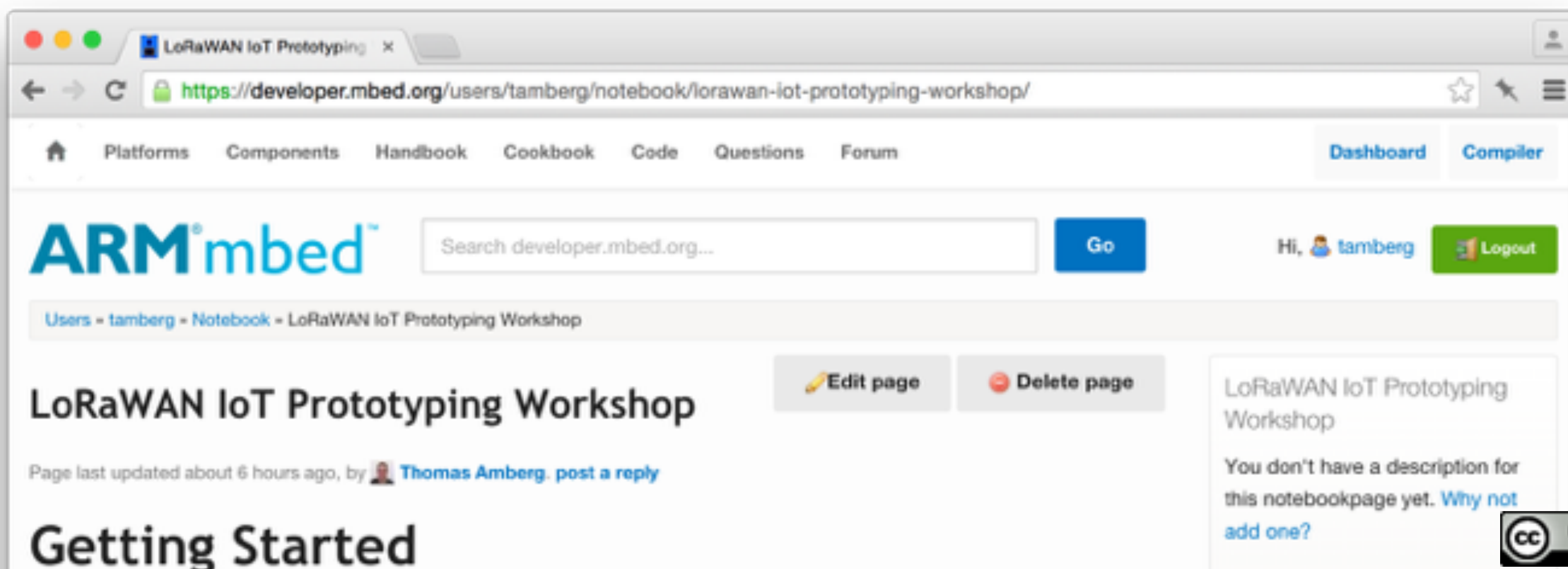
Note: the number of your COM port might differ.

Use it!

Example code is online

The **source code** of the following examples is linked from this notebook, for quick access:

https://developer.mbed.org/users/tamberg/notebook/lorawan-iot-prototyping-workshop/



2) Using sensors and actuators

How to measure and manipulate physical properties with sensors and actuators – the basics of electronics, interactive systems and physical computing, in a few easy examples.

Inputs and outputs

IoT hardware has an **interface to the real world**.

GPIO (**G**eneral **P**urpose **I**nput/**O**utput) pins.

Measure: **read** sensor value from **input** pin
Manipulate: **write** actuator value to **output** pin.

Inputs and outputs can be **digital or analog**.

Inputs and outputs on the Nucleo

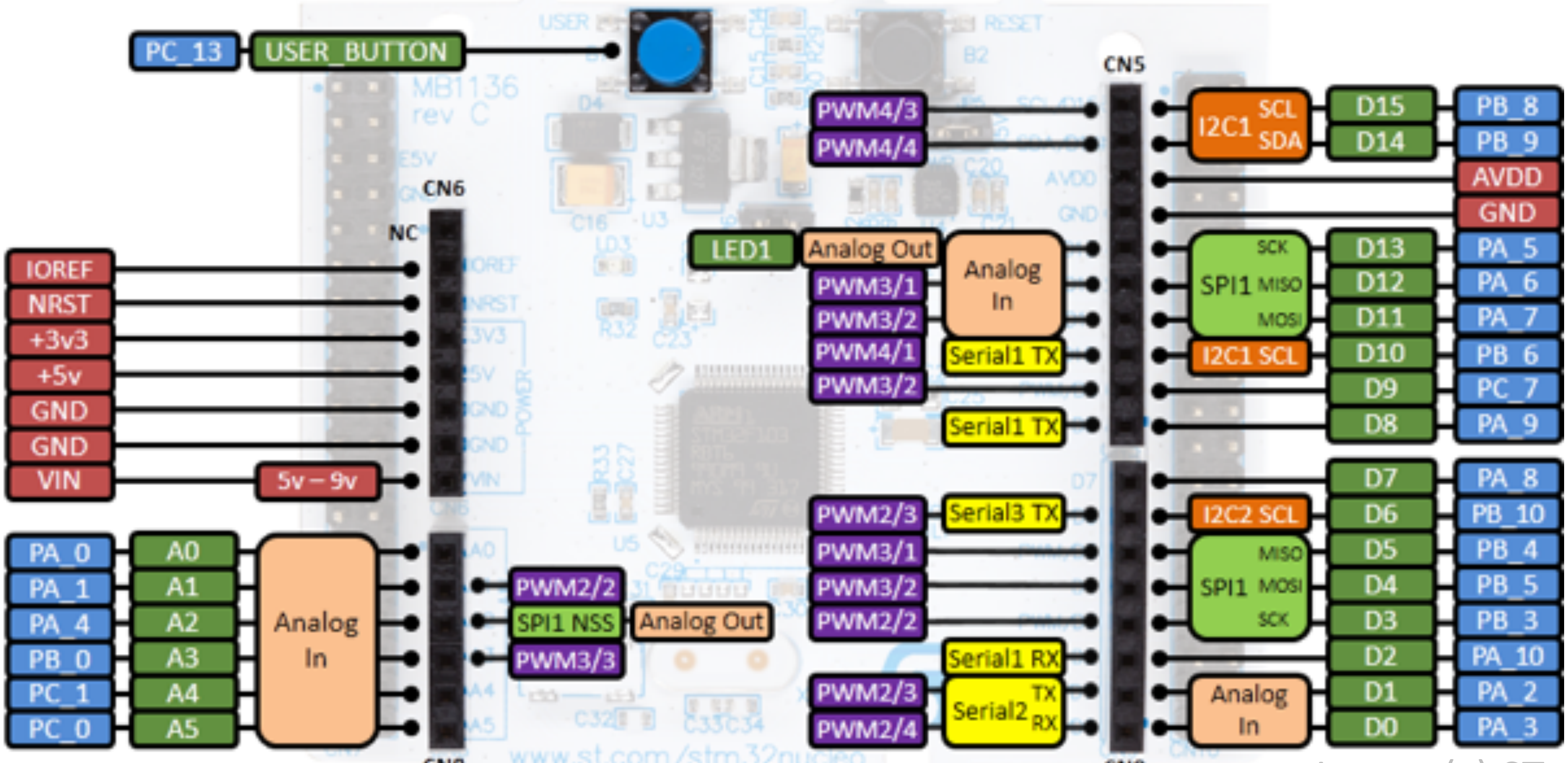
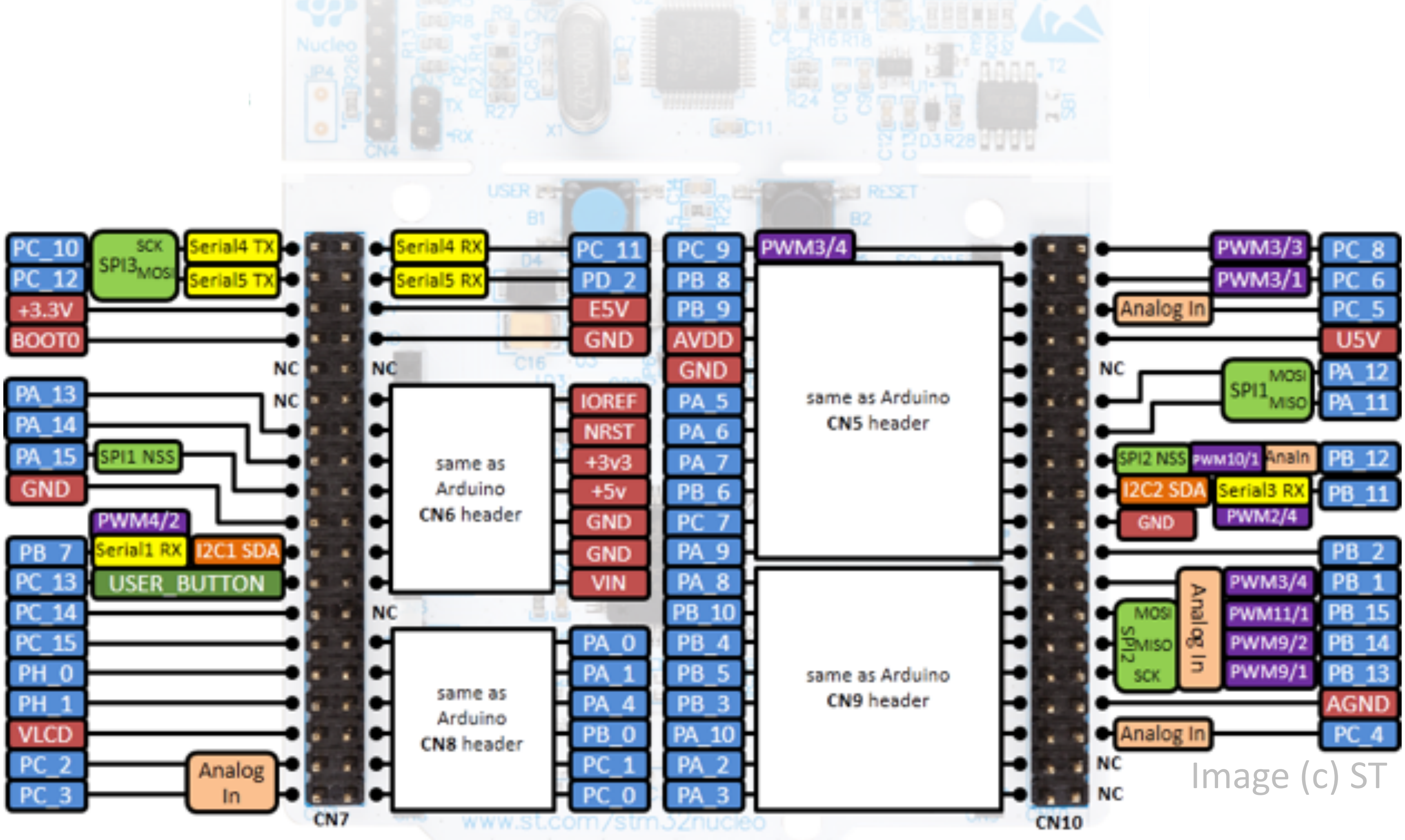


Image (c) ST

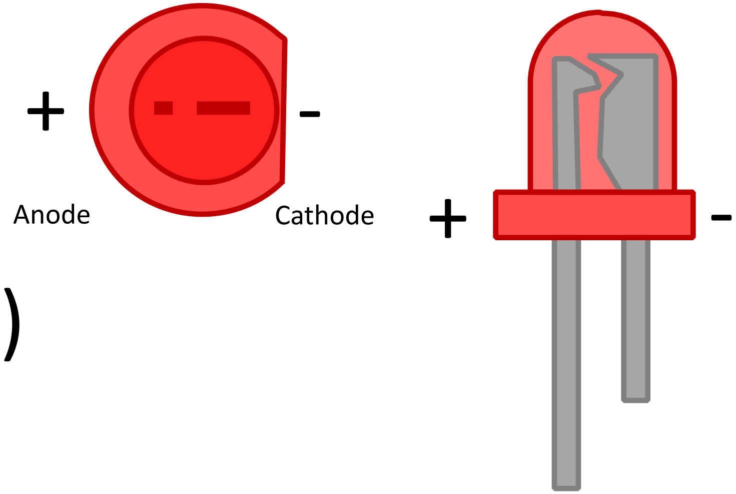
Note: this layout is known as Arduino style GPIOs.

Inputs and outputs on the Nucleo



Note: the Arduino GPIOs are replicated, for better access.

The LED



The **LED** (**L**ight **E**mitting **D**iode) is a simple digital **actuator**.

LEDs have a **short leg** (**-**) and a **long leg** (**+**) and it matters how they are oriented in a circuit.

To prevent damage, LEDs are used together with a 1KΩ **resistor** (or anything from 300Ω to 2KΩ).

The resistor

Resistors are the **workhorse of electronics**.

Resistance is **measured in Ω** (Ohm).

A resistors orientation does not matter.

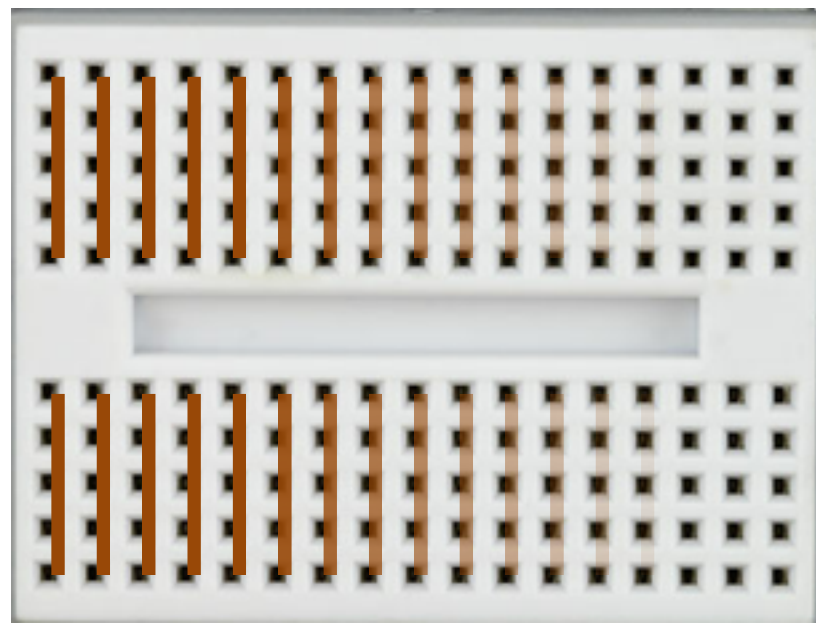
A resistors Ω value is **color-coded** right on it.

Note: color codes are great, but it’s easier to use a multi-meter if you’ve got one, and just measure Ω

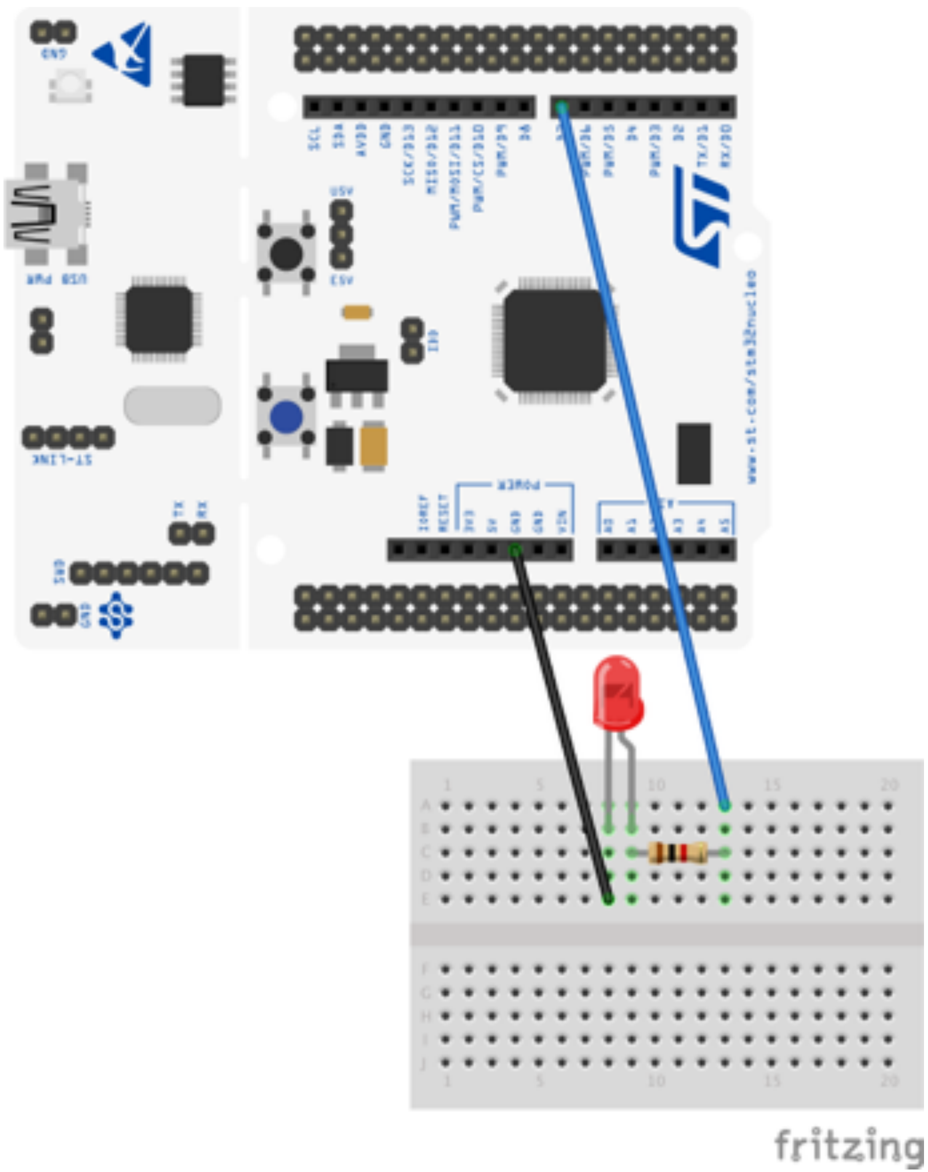
The breadboard

A breadboard lets you wire electronic components without any soldering.

Its holes are connected "under the hood" as shown here.



Wiring a LED with Nucleo



Note: the additional **1K Ω** resistor should be used to prevent damage to the pins / LED if it's reversed

The long leg of the LED is connected to **pin D7**, the short leg to ground (**GND**)

Blinking a LED (digital output)

Nucleo
_blink_led

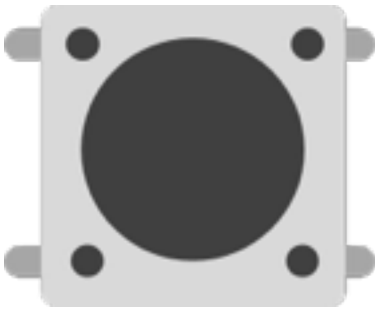
```
#include "mbed.h"
DigitalOut led (D7);

int main() {
   while(1) { // loop
      led.write(1); // on
      wait_ms(500);
      led.write(0); // off
      wait_ms(500);
   }
}
```

Note: blinking a LED is the *Hello World* of embedded software

Set *led* pin as wired in your LED circuit

The switch



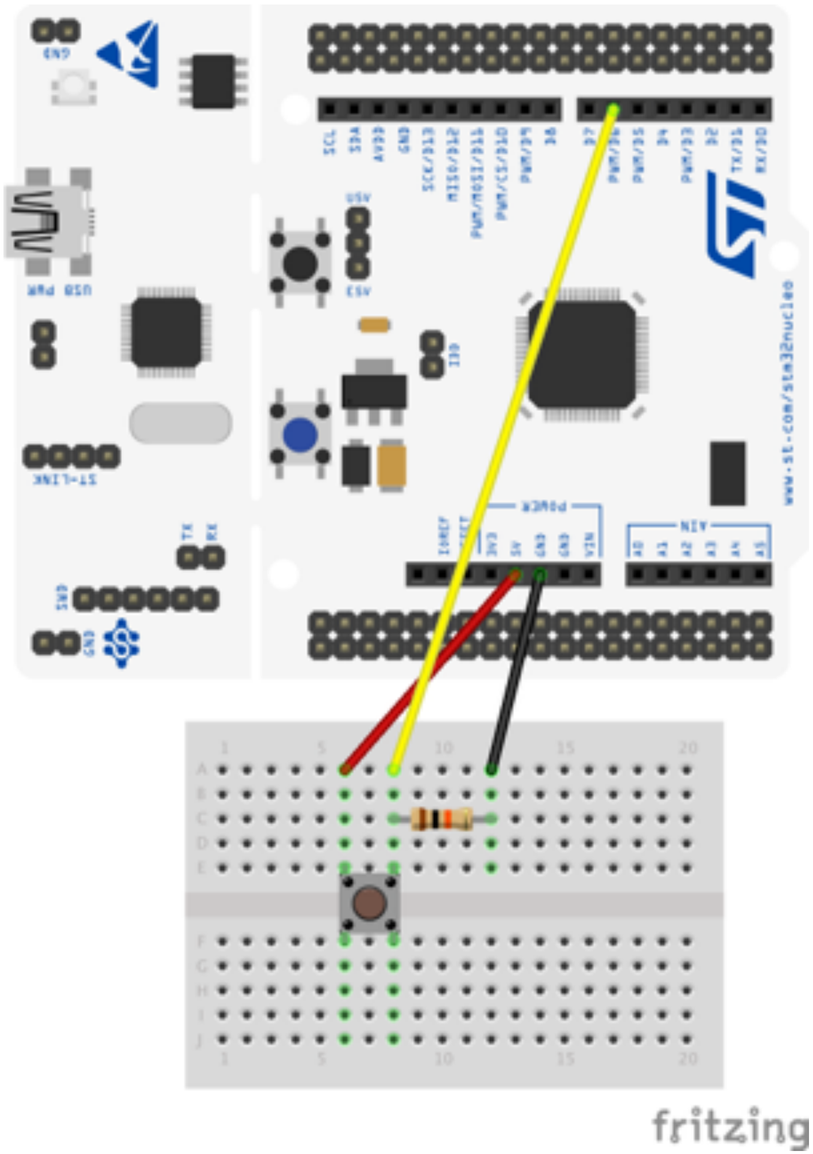
A switch is a simple, digital **sensor**.

Switches come in different forms, but all of them in some way **open** or **close** a gap in a wire.

The **pushbutton** switch has four legs for easier mounting, but only two of them are needed.

Note: you can also easily build your own switches, for inspiration see e.g. http://vimeo.com/2286673

Wiring a switch with Nucleo



Note: the resistor in this setup is called *pull-down* ‘cause it pulls the pin voltage down to GND (0V) if the switch is open

Pushbutton **switch**, **10K Ω** resistor, red wire to **5V**, black wire to **GND**, yellow wire to **D6**

Reading a switch (digital input)

Nucleo_read_switch

```
#include "mbed.h"
DigitalIn button(D6);

int main() {
    while(1) { // loop
        int state = button.read();
        printf("%i\r\n", state); // 0 or 1
        wait_ms(100);
    }
}
```

Note: Open a terminal to see the serial output

Switching a LED

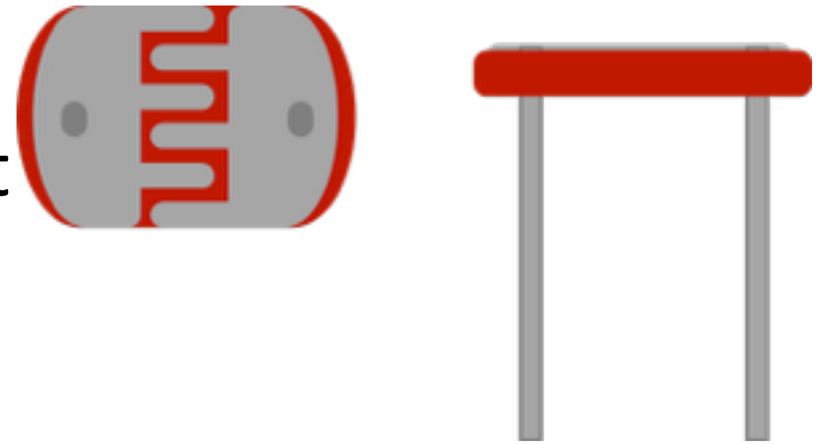
Nucleo
_switch_led

```
#include "mbed.h"
DigitalIn button(D6);
DigitalOut led(D7);

int main() {
    while(1) { // loop
        int state = button.read();
        led.write(state); // 0 or 1
    }
}
```

Note: figure out the wiring from the LED and button example

The LDR

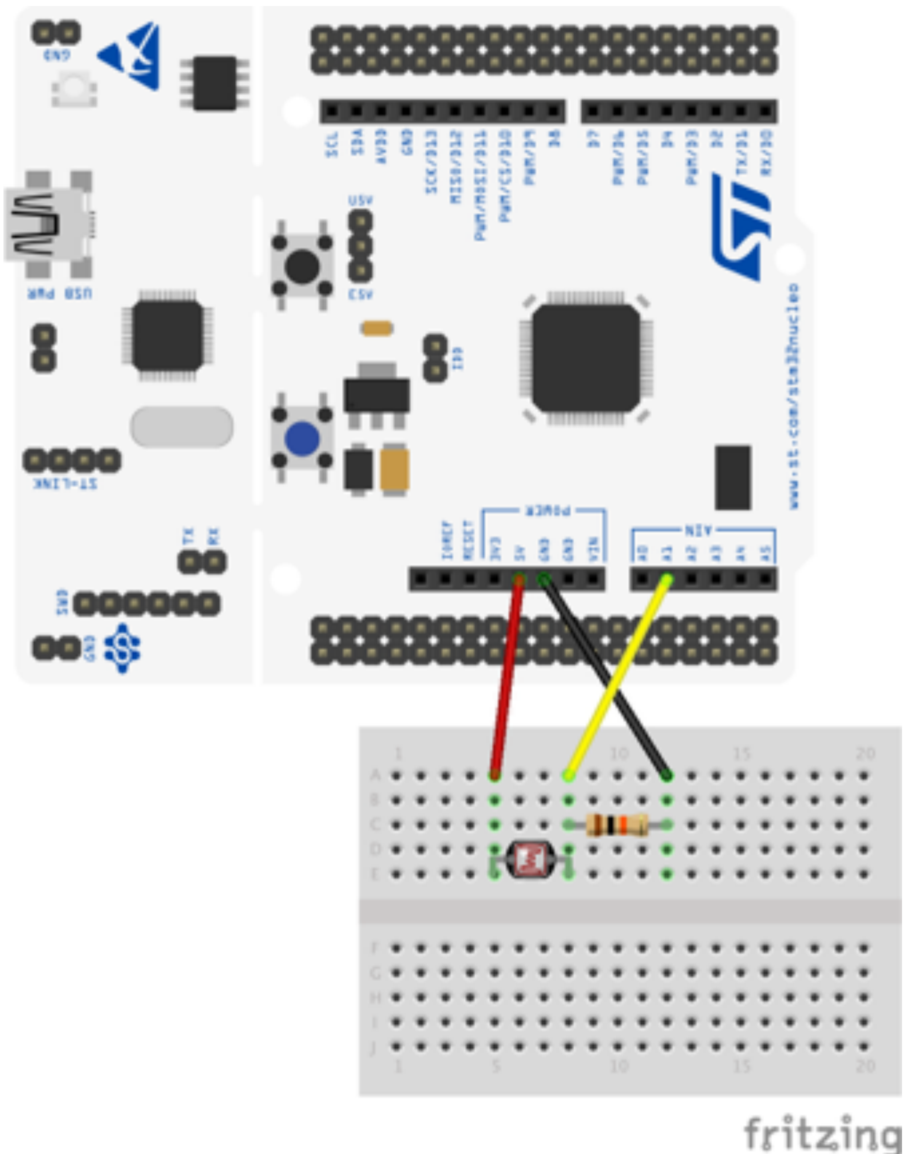


A photoresistor or **LDR** (**l**ight **d**ependent **r**esistor) is a resistor whose resistance depends on light intensity

An LDR can be used as a simple, **analog sensor**

The orientation of an LDR does not matter

Wiring an LDR with Nucleo



Note: this setup is a *voltage-divider, as* the 5V total voltage is divided between LDR and resistor to keep 0V < **A1** < 2.5V

Photoresistor (LDR), **10K Ω** resistor, red wire to **5V**, black wire to **GND**, yellow wire to **A1**

Reading an LDR (analog input)

Nucleo_read_ldr

```
#include "mbed.h"
AnalogIn sensor(A1);

int main() {
  while(1) { // loop
    float value = sensor.read();
    printf("%.2f\r\n", value); // 0.0 to 1.0
    wait_ms(100);
  }
}
```

Open the IDE serial monitor or terminal to see log output

Same code works for other sensors, on pin A1, A2 or A5, if LoRa shield is on.

Note: use e.g. Excel to visualize values over time

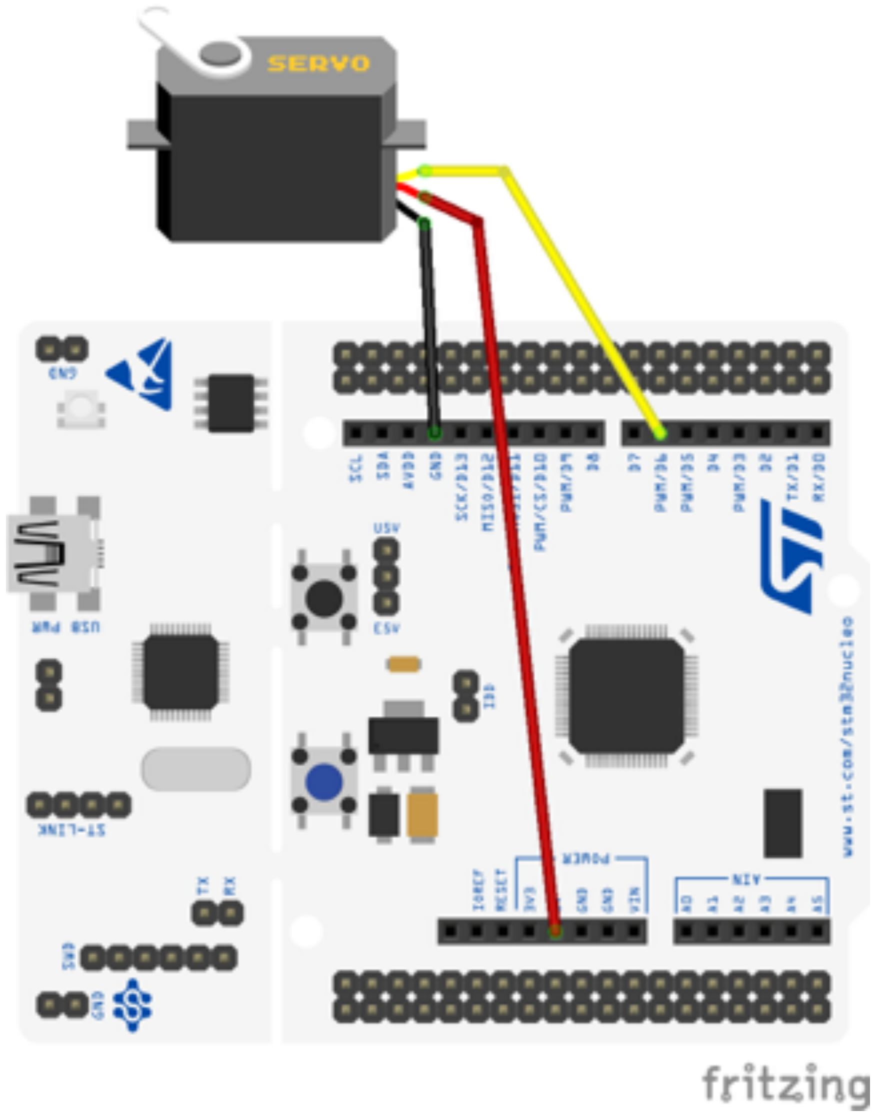
The servo

A **servo** motor takes an input voltage which is translated into a motor position from 0 to 180 degrees.

A servo is a **analog actuator**

To create an analog output for the servo, the device uses pulse width modulation (**PWM**)

Wiring a servo with Nucleo



Note: PWM pin D6 is safe to use with the LoRa shield on.

Red wire to **5V**, black wire to **GND**, yellow wire to **D6**

Controlling a servo (PWM output)

Nucleo_servo

```
#include "mbed.h"
#include "Servo.h" // add library
Servo myservo(D6);

int main() {
  while (1) { // loop
    for (float pos = 0; pos < 1.0; pos += 0.1) {
      myservo.write(pos);
      wait_ms(200);
    }
  }
}
```

Note: *Servo* library lets you use Servos without PWM skills

The *for* loop repeats from pos 0 until pos is 1.0, in steps of 0.1

Controlling a servo with an LDR

Nucleo _servo_ldr

```
#include "mbed.h"
#include "Servo.h" // add library
AnalogIn sensor(A1);
Servo myservo(D6);

int main() {
  while (1) { // loop
    float value = sensor.read();
    myservo.write(value);
    wait_ms(200);
  }
}
```

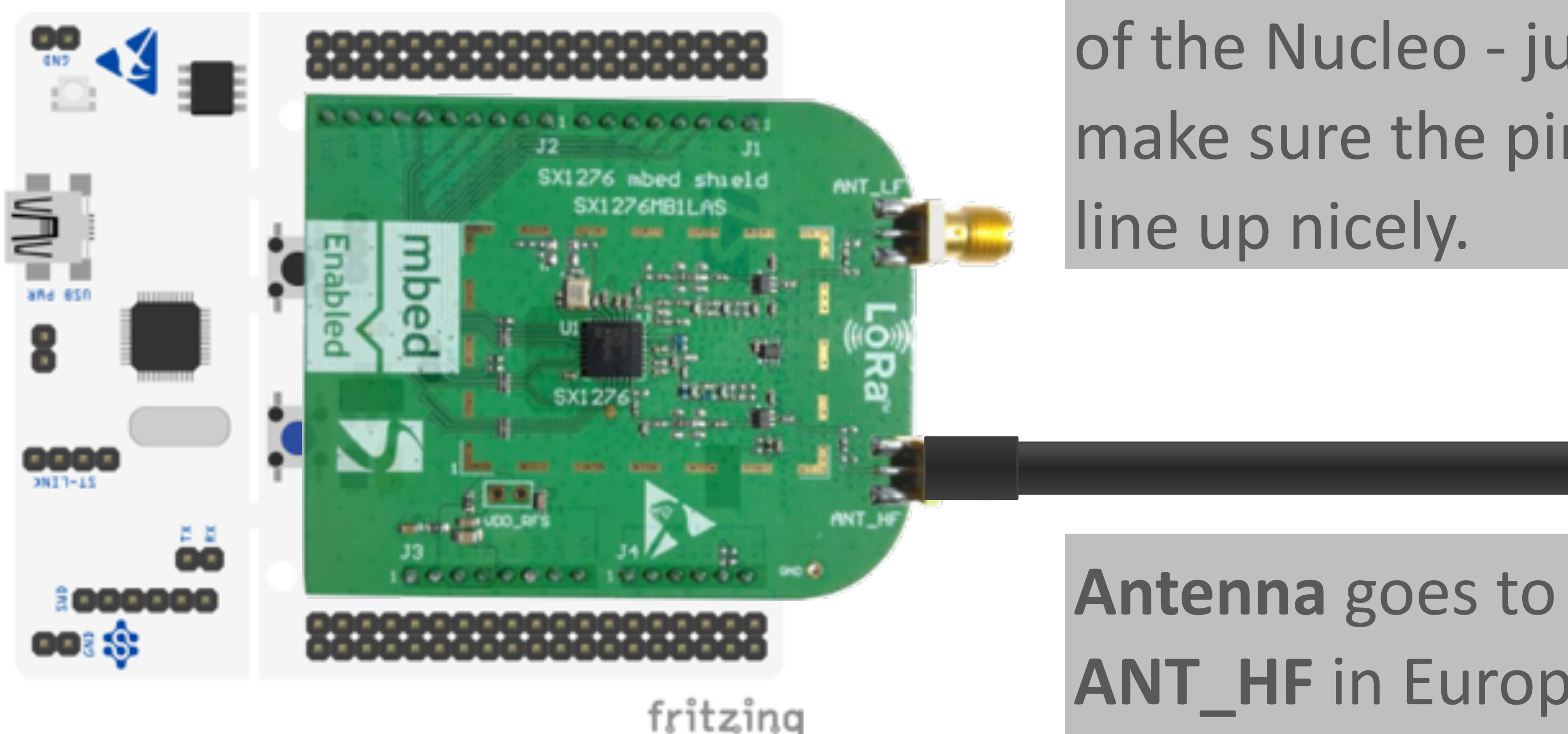
Note: combine the wiring diagrams of both, Servo & LDR

Value has the right range for a servo, as both are analog 0..1

3) Connecting to LoRaWAN

How to connect your Mbed device to the LoRa wide area network and transfer custom data to the ThingPark LoRa platform.

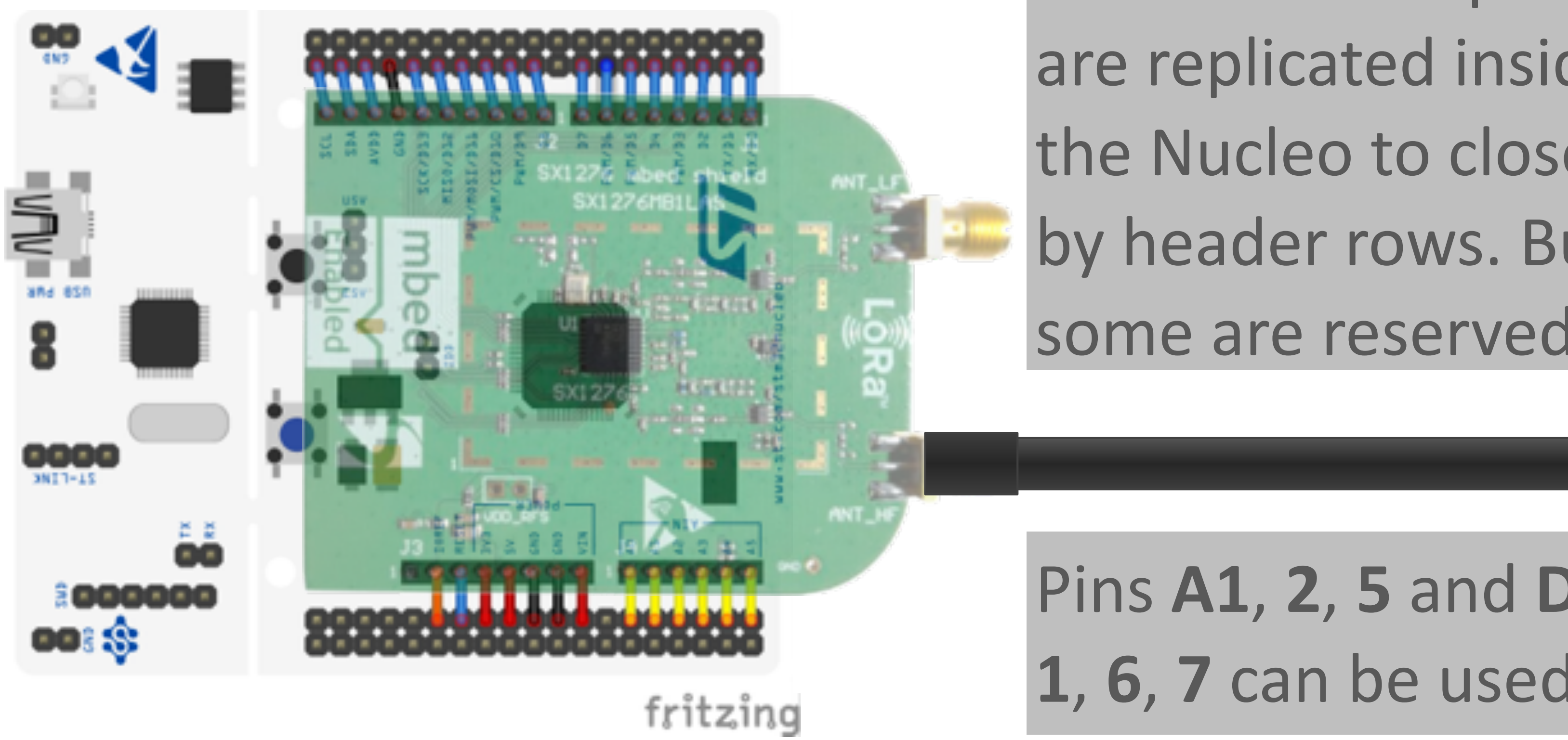
Adding the LoRa shield



Note: the LoRa shield goes on top of the Nucleo - just make sure the pins line up nicely.

Antenna goes to **ANT_HF** in Europe.

Inputs and outputs with a shield



Note: Arduino pins are replicated inside the Nucleo to close-by header rows. But some are reserved.

Pins **A1**, **2**, **5** and **D0**, **1**, **6**, **7** can be used.

All previous examples still work, just use F/M wires.

LoRaWAN
_send_text

Sending text with LoRaWAN

```
#include "mbed.h" ...
void onSendFrame (osjob_t* j) {
  const char* message = "Hello"; // ASCII only
  int frameLength = strlen(message); // keep it < 32
  for (int i = 0; i < frameLength; i++) {
    LMIC.frame[i] = message[i];
  }
  int result = LMIC_setTxData2(
    LORAWAN_APP_PORT, LMIC.frame, frameLength,
    LORAWAN_CONFIRMED_MSG_ON);
}
...
```

No worries, details do not matter here.

This code copies the message into a LoRa frame and sends it.

Note: this won't work, to **set your keys** see next slide.

Setting your keys in the code

LoRaWAN_send_text

Search for *TODO* in main.cpp

```
// TODO: enter your ...
#define LORAWAN_DEV_ADDR (uint32_t) 0x01234567
static uint8_t NwkSKey[] = { // TODO: enter your key,
    // e.g. for 00112233445566778899AABBCCDDEEFF
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF
}; ...
static uint8_t ArtSKey[] = { // TODO: enter your key
    0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??,
    0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??
};
...
```

The **0x**.. is important.

Same here

Note: keys change **per device**, **ask** us to get them.

LoRaWAN _send_text

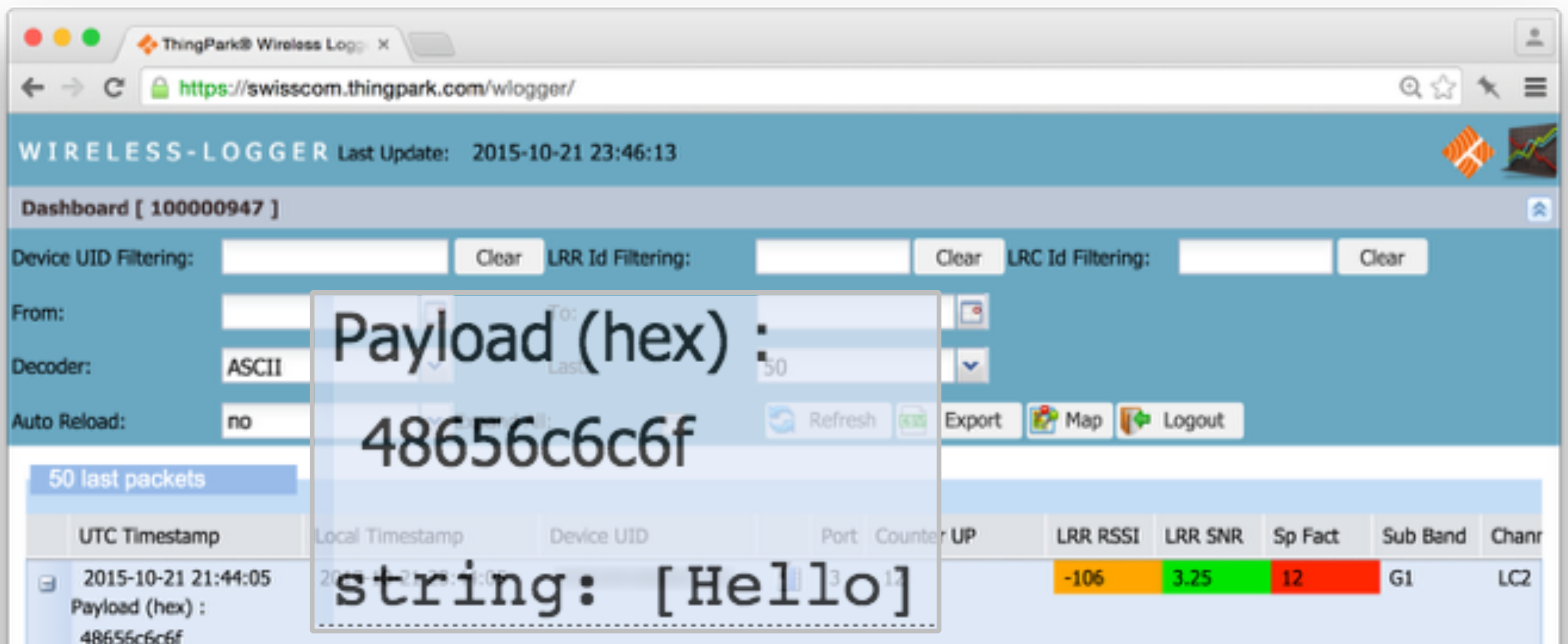
Sending text (Nucleo output)

Run the code and open a terminal to check the serial output. Each **TXCOMPLETE** means a LoRa packet was sent from the Nucleo, received by a LoRaWAN gateway and confirmed.

Note: the second TXCOMPLETE takes **minutes** to arrive.

Sending text (ThingPark logger)

Use the credentials ThingPark sent you to log in at https://swisscom.thingpark.com/wlogger/



Note: set decoder ASCII, **refresh** to see packets, expand.

Bonus for programmers

You just sent your first LoRaWAN packets.

Try sending an **int** or **float** instead of a string.

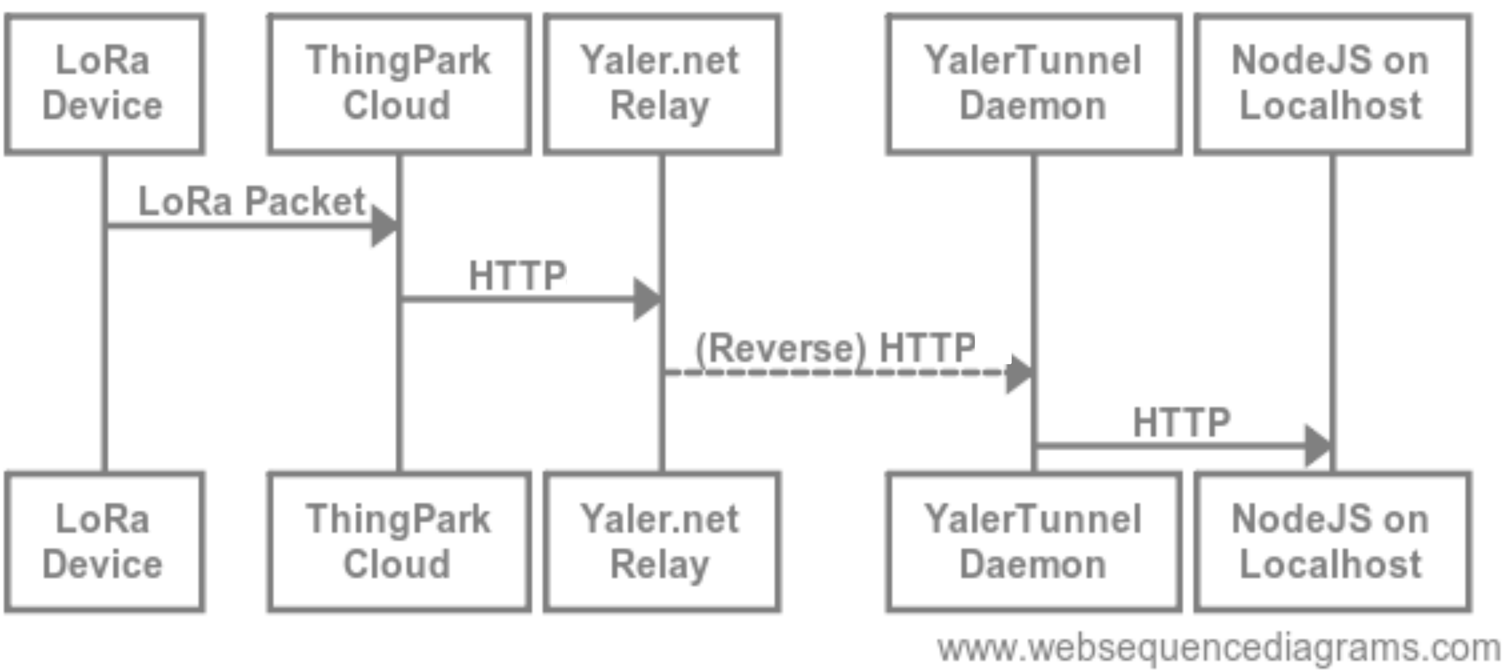
There is no decoder on the other end, though.

But we'll get to the raw bytes in the next part.

Note: convert values to bytes, beware of endianness.

4) Running a Web service

How to run a simple NodeJS Web service and make it accessible via the Yaler relay to receive data from the ThingPark LoRa platform.



Installing NodeJS on your Mac / PC

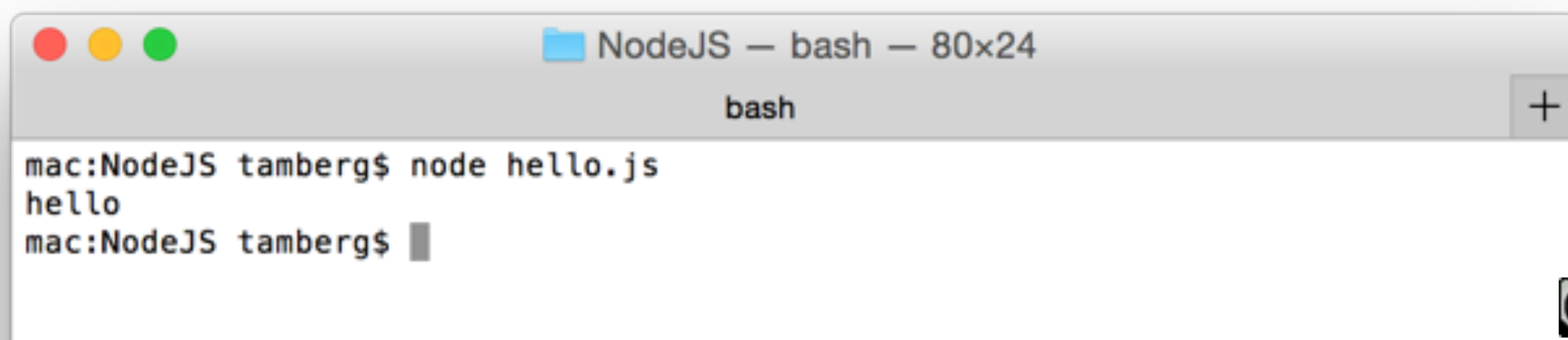
Install NodeJS from https://nodejs.org/en/

Create a text file named **hello.js**, and enter

console.log("hello");

Open a **terminal** at the same location and type

$ *node hello.js*



Running HTTP Logger on localhost

Download the **http-logger.js** example from
https://bitbucket.org/tamberg/iotworkshop/raw/tip/NodeJS/http-logger.js

Open a **terminal** at the same location and type

$ *node http-logger.js*

Access http://127.0.0.1:8080/ on your computer

Note: terminal now shows the raw browser request.

Enabling remote access to localhost

Download and unzip YalerTunnel.src.zip from https://bitbucket.org/yaler/yalertunnel/downloads/YalerTunnel.src.zip

Open a **terminal** at the same location and type

$ *javac YalerTunnel.java*

Get a free **relay domain** at https://yaler.net/ , then

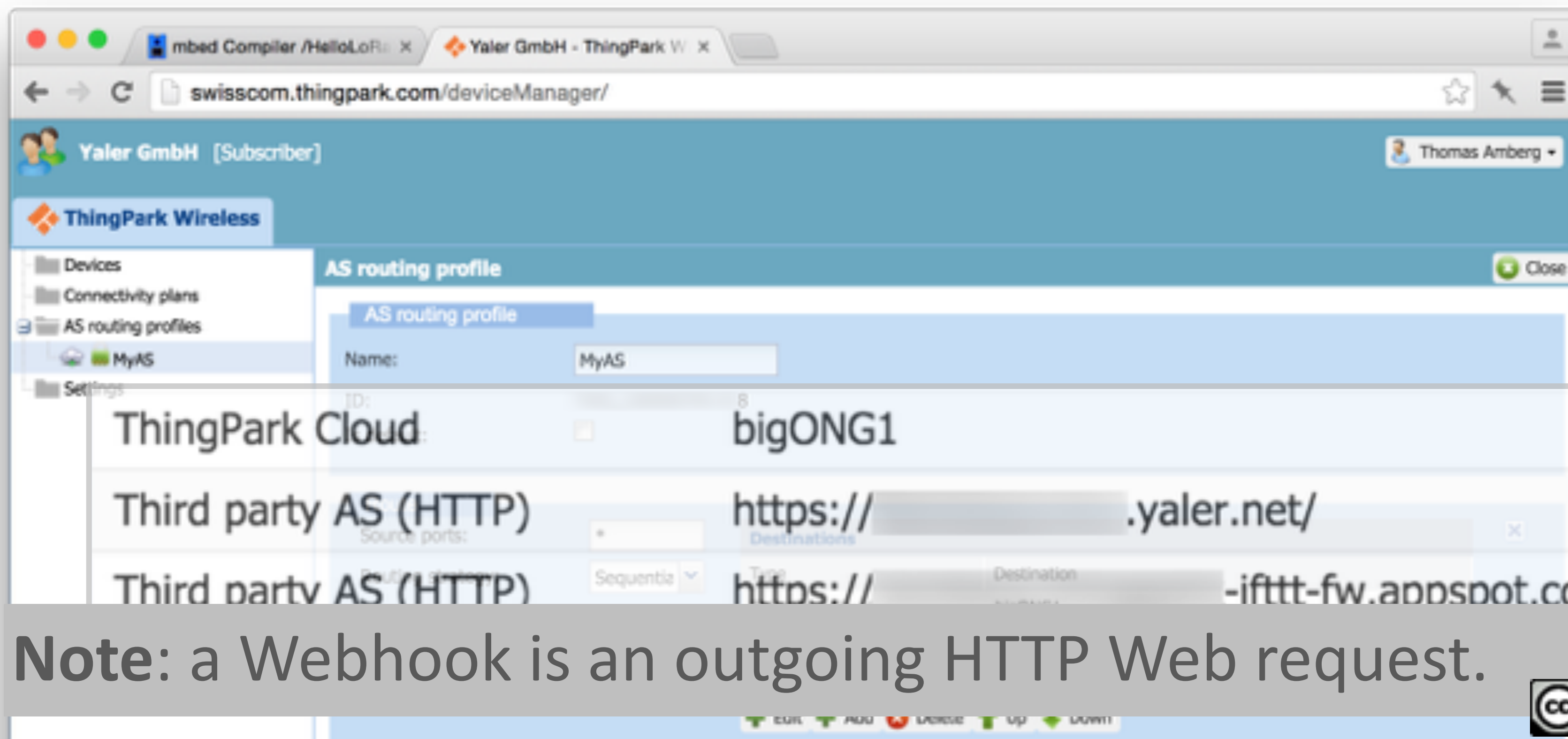
$ *java YalerTunnel server 127.0.0.1:8080 try.yaler.net:80 RELAY_DOMAIN*

Access http://RELAY_DOMAIN.yaler.net/

Setting up a Webhook in ThingPark

http://swisscom.thingpark.com/deviceManager/

Go to ***AS routing profiles*** > *Edit* > *Add* your URL



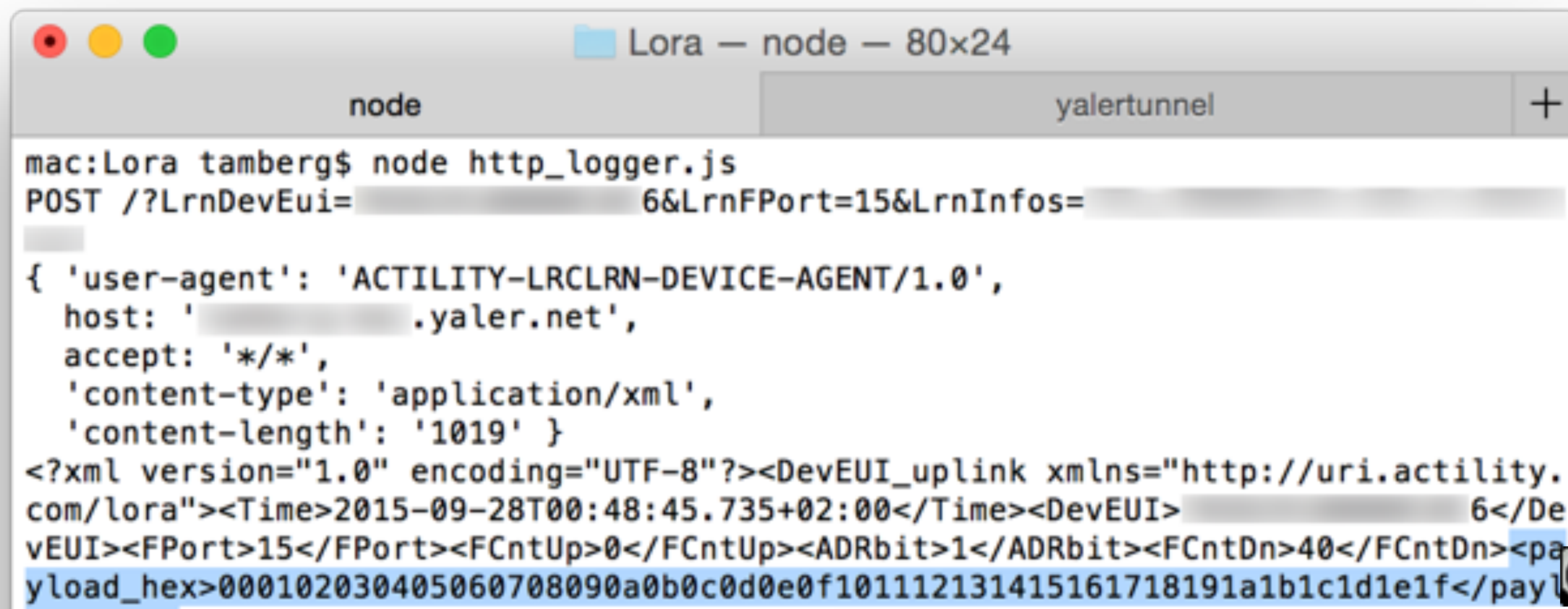
ThingPark Cloud	bigONG1
Third party AS (HTTP)	https:// .yaler.net/
Third party AS (HTTP)	https:// -ifttt-fw.appspot.com/

Note: a Webhook is an outgoing HTTP Web request.

HTTP Logger output from ThingPark

Restart the LoRaWAN example using the black reset button on the **Nucleo** to **trigger a packet**.

Check the **http_logger.js** output, should be XML.



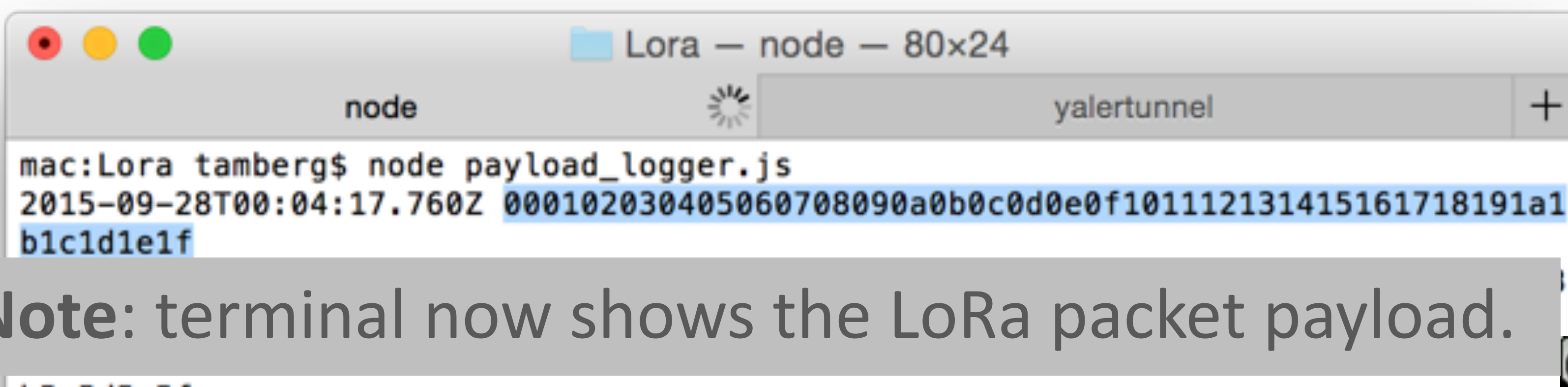
Running ThingPark Logger

Download the **thingpark-logger.js** example from https://bitbucket.org/tamberg/iotworkshop/raw/tip/NodeJS/thingpark-logger.js

Open a **terminal** at the same location and type

$ *node thingpark-logger.js*

Keep the yalertunnel daemon running as before.



Note: terminal now shows the LoRa packet payload.

5) Storing sensor data

How to store measurements in the ThingSpeak sensor data repository and display charts.

Post to ThingSpeak with Curl

The ThingSpeak service lets you store, **monitor** and share **sensor data** in open formats. Sign up at https://thingspeak.com/ to create a channel and get API keys, then try the following:

$ **curl** -vX **POST** http://api.thingspeak.com/**update**?key=*WRITE_API_KEY*&field1=42

$ **curl** -v http://api.thingspeak.com/channels/*CHANNEL_ID*/**feed.json**?key=*READ_API_KEY*

Running ThingSpeak Forwarder

Download the ...**-thingspeak-forwarder.js** example
https://bitbucket.org/tamberg/iotworkshop/raw/tip/NodeJS/thingpark-thingspeak-forwarder.js

Open a **terminal** at the same location and type

$ *node thingpark-thingspeak-forwarder.js*

Keep the yalertunnel daemon running as before.

Trigger a LoRa packet and check the result at

https://thingspeak.com/channels/CHANNEL_ID

Note: set your WRITE_API_KEY in the .js source code.

6) Controlling your device

Not yet

How to send control data to the ThingPark LoRa platform with Curl and the IFTTT Do Button App.

Post to ThingPark with Curl

Not yet

The ThingPark service lets you send data to the device. Try the following:

~~$ **curl** -vX **POST** http://api.thingpark.com/**update**?key=*WRITE_API_KEY*&field1=42~~

IFTTT Do Button with LoRaWAN

Not yet

Connect the Maker Channel at https://ifttt.com/maker

Get the **Do Button App**, tap *'+' > Channels > Maker > Create a new recipe > Make a Web request >* ... then go to https://ifttt.com/myrecipes/do for convenience

~~URL: http://api.thingpark.com/led?color=330033~~

~~Method: POST~~

~~Content Type: application/x-www-form-urlencoded~~

7) Mash-ups with 3rd party services

How to forward data to IFTTT and create mash-up recipes to integrate 3rd party services.

IFTTT

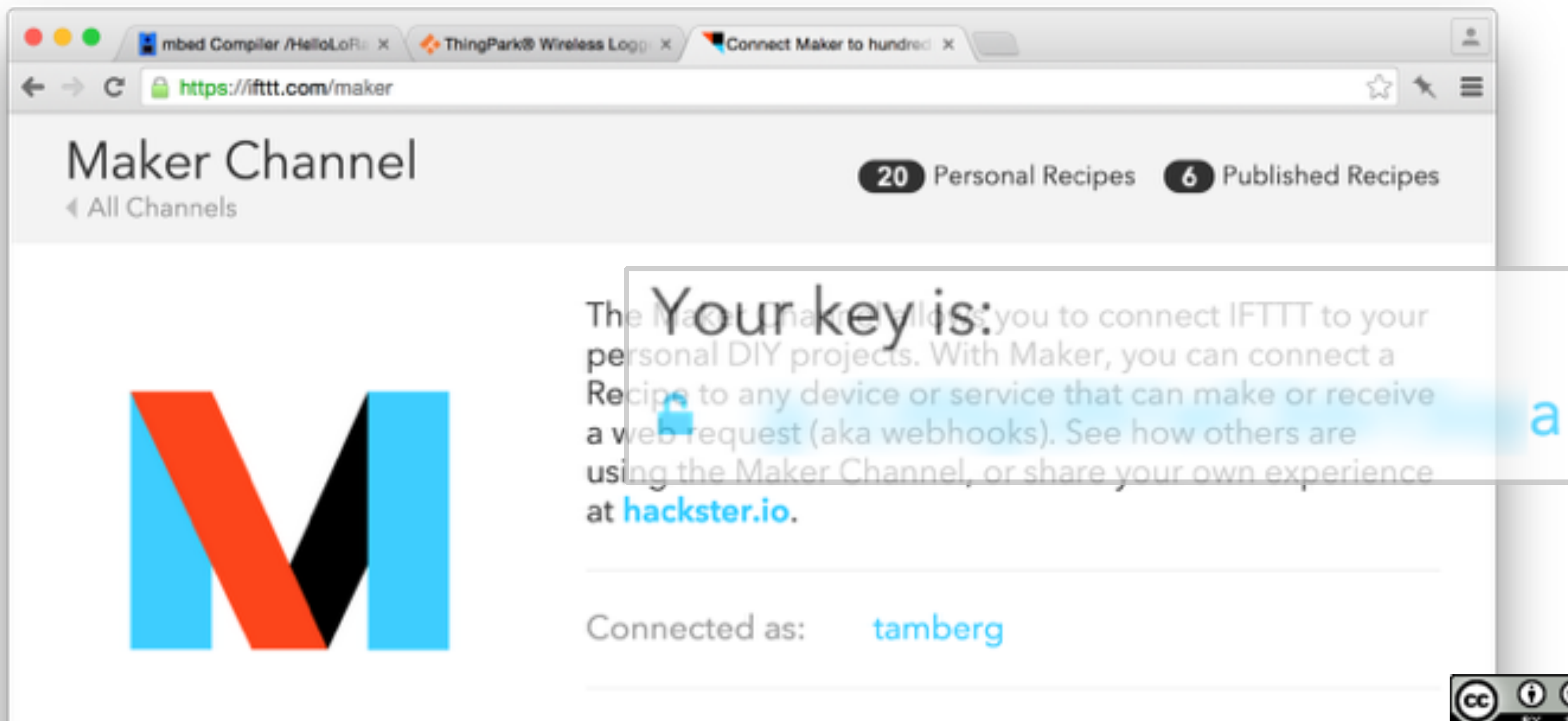
If This Then That (IFTTT) is a **mash-up** platform

An IFTTT Recipe connects two Web services (or a service and a device) using their Web **API**s

The IFTTT **Maker Channel** uses **Webhooks** (outgoing HTTP requests) to call your device, and you can use **Web requests** to trigger IFTTT.

IFTTT Maker Channel

Sign up at https://ifttt.com/ and get your MAKER_CHANNEL_KEY at https://ifttt.com/maker



Running IFTTT Forwarder

Download the **thingpark-ifttt-forwarder.js** code
https://bitbucket.org/tamberg/iotworkshop/raw/tip/NodeJS/thingpark-ifttt-forwarder.js

Open a **terminal** at the same location and type

$ *node thingpark-ifttt-forwarder.js*

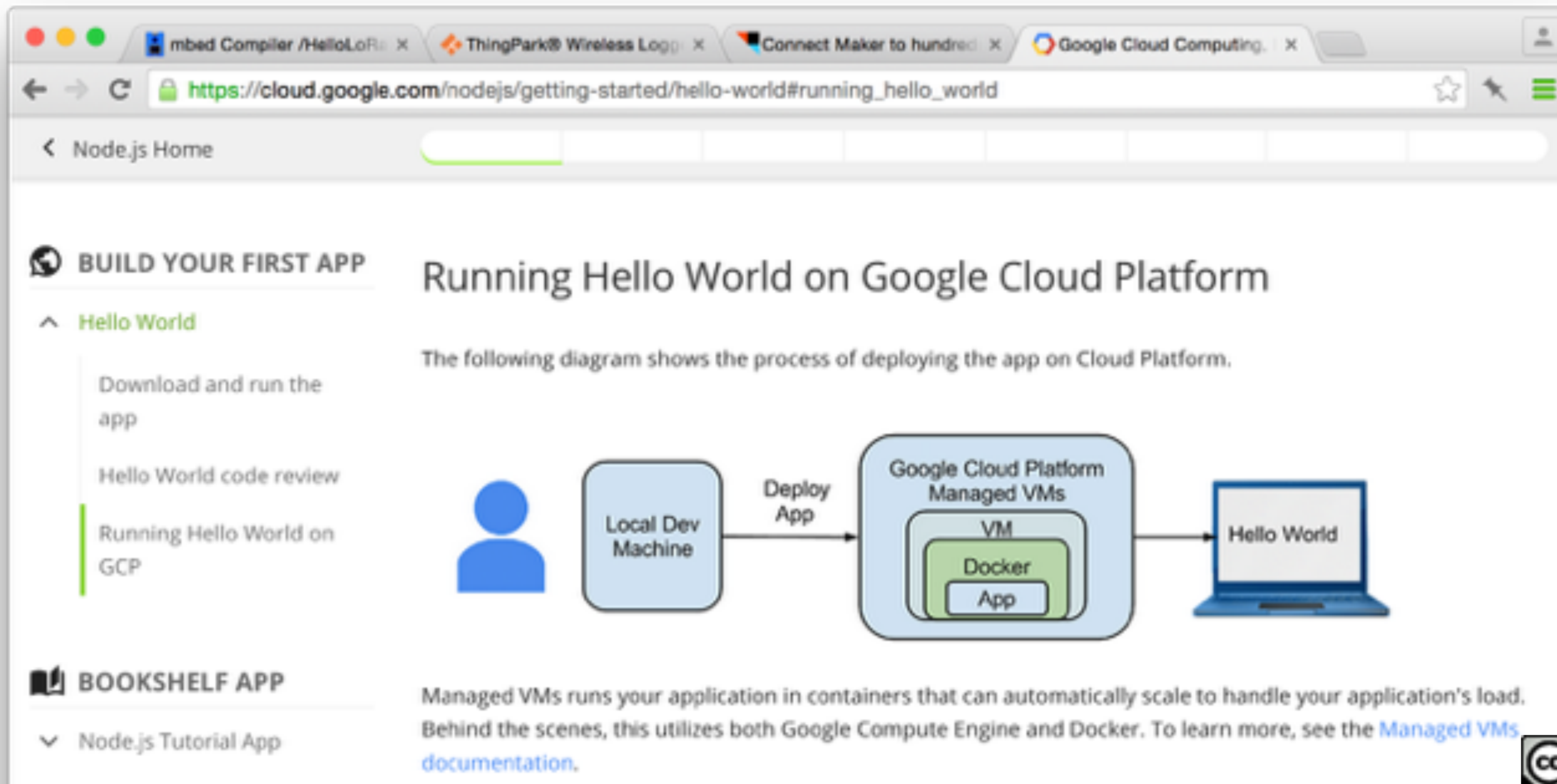
Keep the yalertunnel daemon running as before.

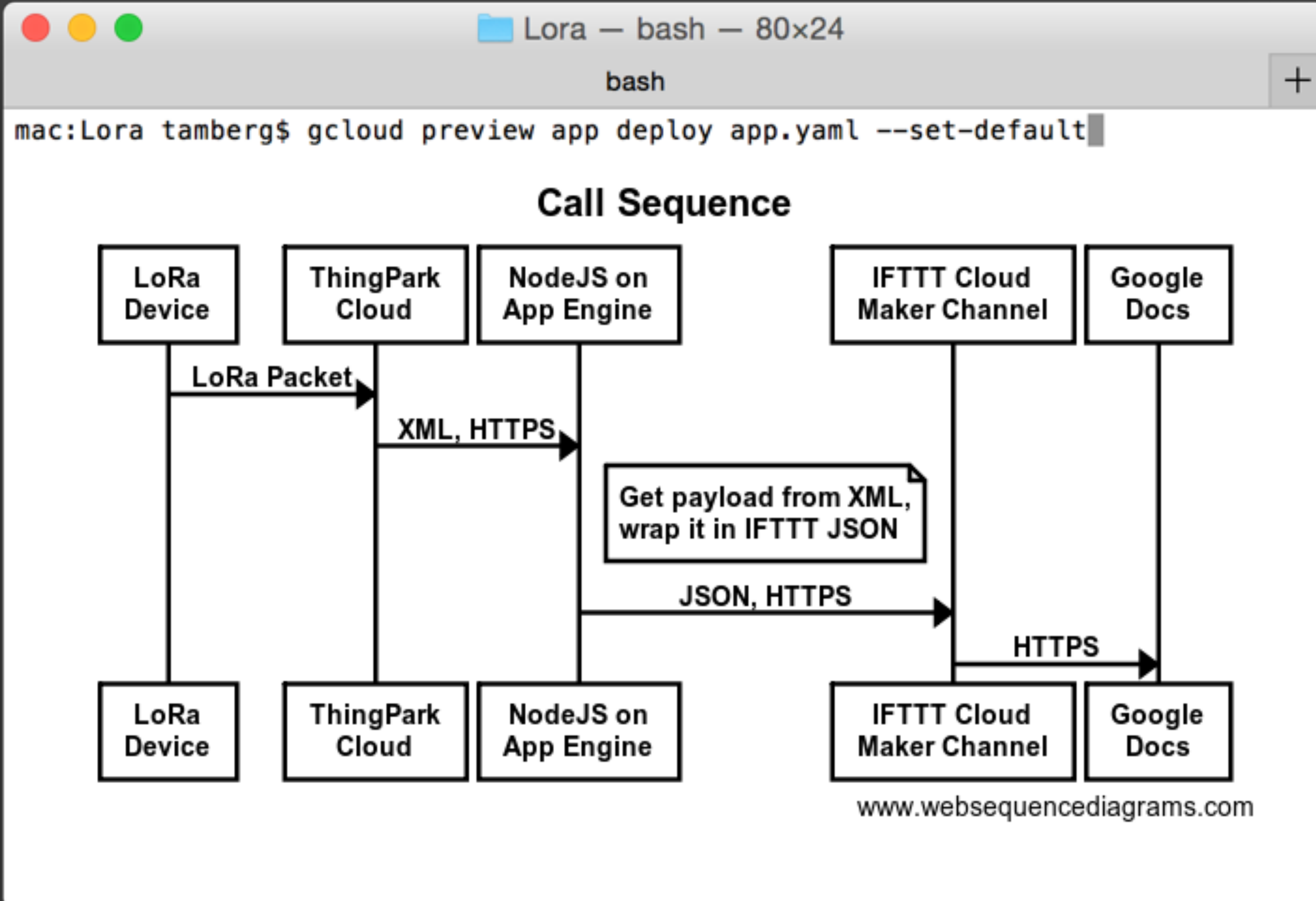
Create a recipe on IFTTT using the Maker channel, e.g. https://ifttt.com/recipes/335901-lorawan-log

Note: make sure to set your MAKER_CHANNEL_KEY.

Running NodeJS in the cloud

https://cloud.google.com/nodejs/getting-started/hello-world shows you how.




Lora — bash — 80×24
bash
mac:Lora tamberg$ gcloud preview app deploy app.yaml --set-default
Call Sequence
LoRa Device
ThingPark Cloud
NodeJS on App Engine
IFTTT Cloud Maker Channel
Google Docs
LoRa Packet
XML, HTTPS
Get payload from XML, wrap it in IFTTT JSON
JSON, HTTPS
HTTPS
LoRa Device
ThingPark Cloud
NodeJS on App Engine
IFTTT Cloud Maker Channel
Google Docs
www.websequencediagrams.com

Thanks.

thomas.amberg@yaler.net

twitter.com/tamberg

yaler.net

Slides at goo.gl/K1GWvz →