

# IoT Engineering

## 9: Dashboards and Apps for Sensor Data

CC BY-SA 4.0, T. Amberg, FHNW  
(unless noted otherwise)

Slides [tmb.gr/iot-9](https://tmb.gr/iot-9)

# Overview

These slides introduce *sensor dashboards and apps*.

From hosted services and apps to "build your own".

How to move data and integrate with platforms.

# Prerequisites

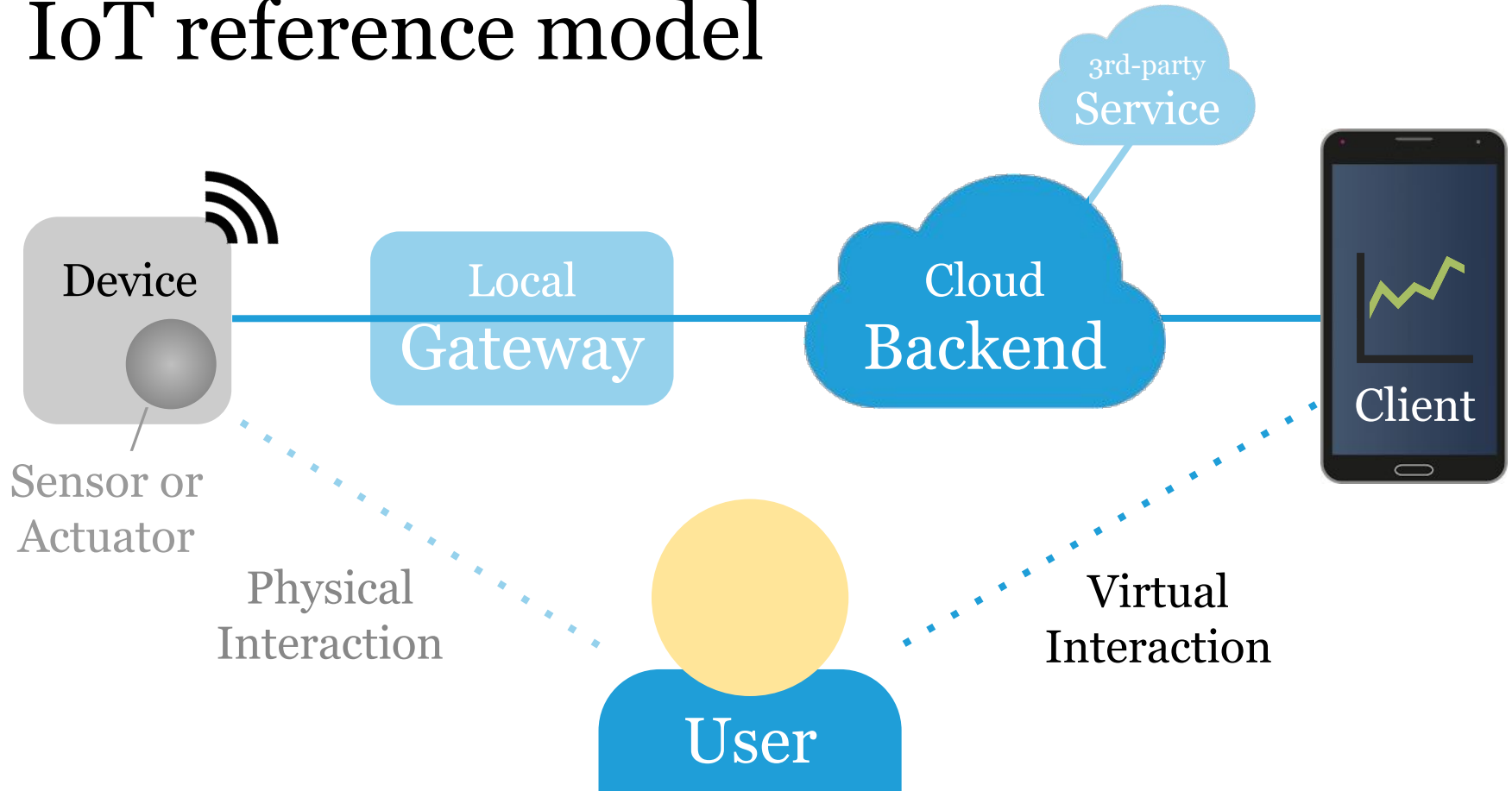
We use [curl](#) and the [mqtt](#) CLI tool to emulate devices.

The [Raspberry Pi](#) with [Node.js](#) will be our "backend".

Some examples require [Docker](#) on your computer.

Note: Docker part is still in beta, will be updated.

# IoT reference model



# Dashboards

Dashboard as a service — easy set up, but: dependency.

Self-hosted dashboard — keep control, but: operations.

Graph libraries — re-use, flexible, but: dev & ops work.

Build your own — max. control, but also max. work.

# Dashboard as a service

Backend, defining data formats & information model.

Device-side backend API to get data in (HTTP/MQTT).

Data storage or caching functionality (sliding window).

Client-side API to get data out (HTTP or Websocket).

Private or public dashboard Web UI or client app.

# Information model

The information model defines how data is structured.

It's the "common denominator" of all involved parties.

Data formats (on the wire) define how it's transported.

The information model is more about data semantics.

E.g. what is a device, what is a sensor measurement? 7

# ThingSpeak

ThingSpeak timestamps, stores and displays data.

It supports per device *channels* with 1-N *fields* each.

Graph controls can be embedded in HTML Web UIs.

ThingSpeak provides HTTP and MQTT endpoints.



# ThingSpeak HTTP API

ThingSpeak has a **device-** and **client-side HTTP API**.

Host: `api.thingspeak.com`

Port: 80 or 443

POST `/update?key=WRITE_API_KEY&field1=42`

GET `/channels/CHANNEL_ID/feed.json?  
key=READ_API_KEY`

# ThingSpeak MQTT API

ThingSpeak has a [device-](#) and [client-side MQTT API](#).

Host: `mqtt.thingspeak.com`

Port: 1883 or 8883 (or Websocket: 80, 443)

```
PUB -t 'channels/CHANNEL_ID/publish/\nWRITE_API_KEY' -m 'field1=42&field2=23'
```

```
SUB -t 'channels/CHANNEL_ID/subscribe/\nFORMAT/READ_API_KEY'
```

# Uniontown Weather Data

Channel ID: 3

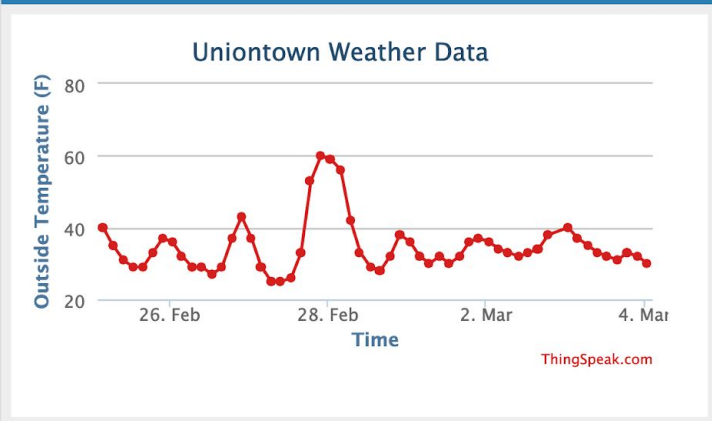
Author: [iothans](#)

Access: Public

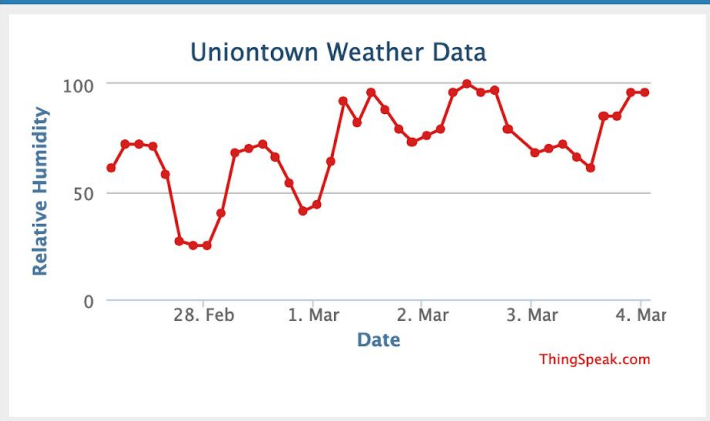
Weather data from Uniontown, PA

🔑 [temperature](#), [humidity](#), [weather station](#), [dew point](#), [channel\\_3](#)

Field 1 Chart



Field 2 Chart



# Cayenne

**Cayenne**\* apps display data from any MQTT broker.

Per *thing* (device), multiple data *channels* are supported, with *type*, *unit* and *value* fields.

SDKs for **ESP8266**, **Node.js**, etc., simplify sending values, e.g. encoded in the **CayenneLPP** data format. Cayenne also provides an MQTT broker endpoint.

\*Announced [end-of-life](#), links might break.

# Cayenne MQTT API

Cayenne specifies a device-side [MQTT API](#) and SDKs.

Host: `mqtt.mydevices.com`

Port: 1883 or 8883

```
PUB -t 'v1/MQTT_USER/things/DEVICE_ID/\n      data/CHANNEL_ID' -m 'TYPE,UNIT=VALUE'
```

For details, see the Cayenne [payload documentation](#), which is wrapped in device specific integrations.

# Cayenne HTTP API

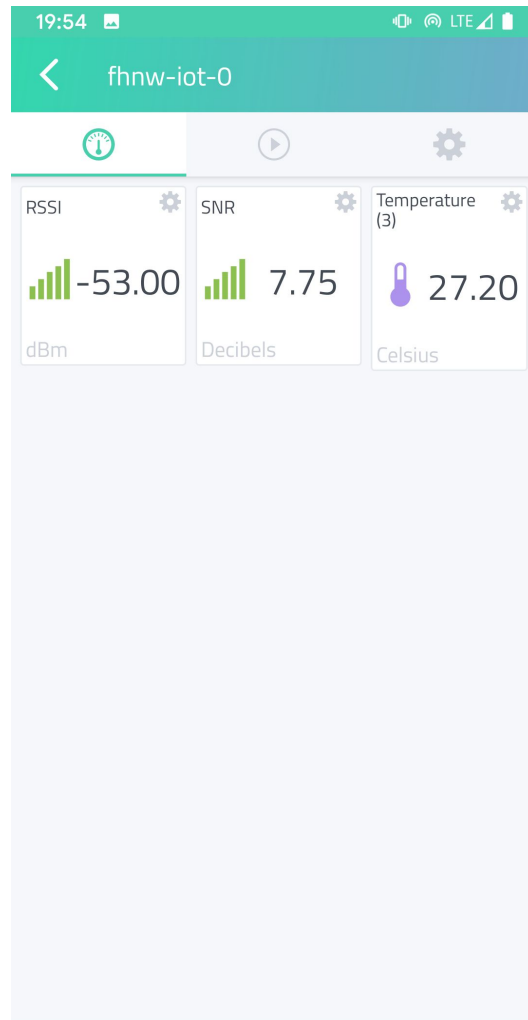
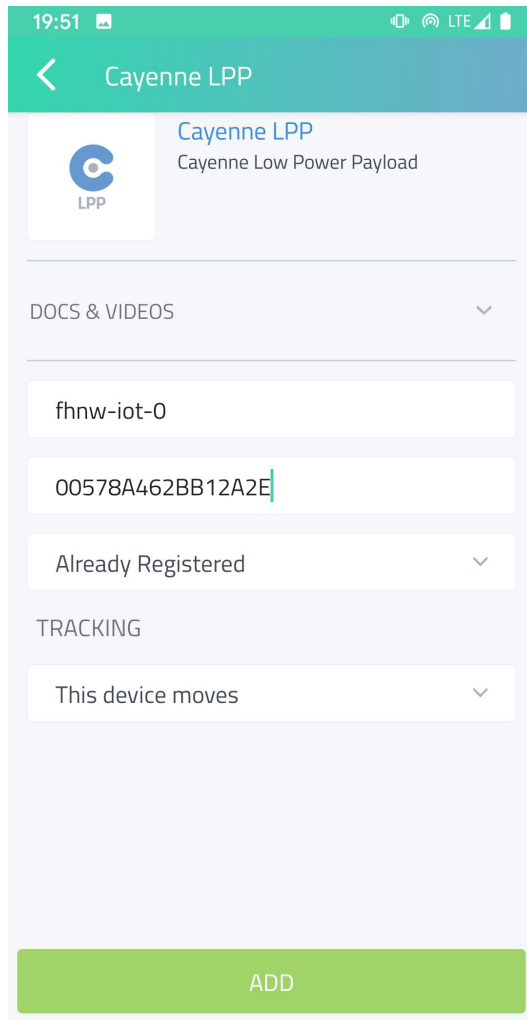
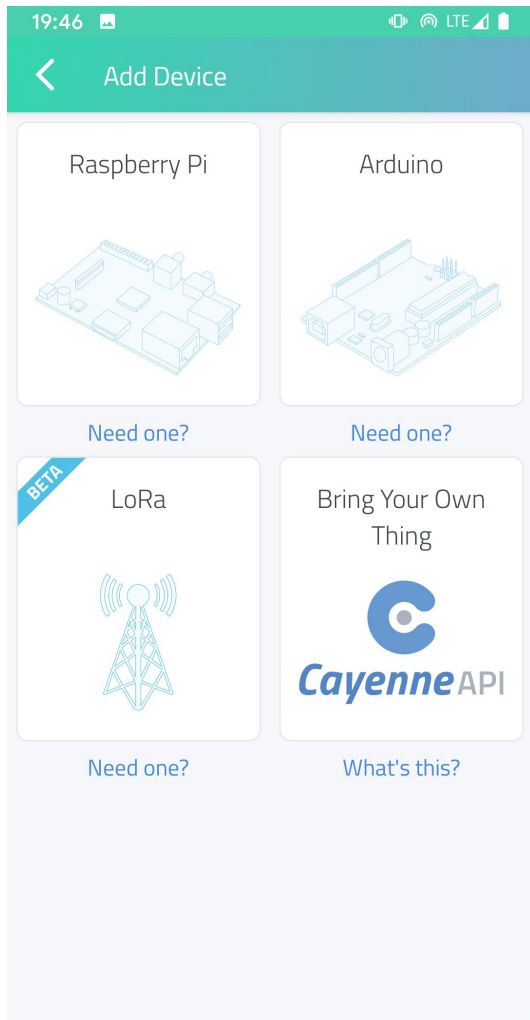
The Cayenne **iOS** and **Android** apps use an **HTTP API**.

It enables device management, to set up devices/keys.

It is client-only, devices connect to the MQTT API.

As soon as data comes in, graphs are generated\*.

\*The data format includes types, units and values.



# Integrations

Integrations allow backend services to work together.

E.g. TTN backend integration w/ Cayenne MyDevices.

Integration adapters can be provided by either party.

The adapter code can be hosted on either backend\*.

Settings include endpoint, format and API tokens.

\*Or somewhere else, if you use custom glue code.





Overview

End devices

Live data

Payload formatters

Integrations

MQTT

Webhooks

Storage Integration

AWS IoT

Azure IoT Hub

LoRa Cloud

Collaborators

## Template information

Drag-and-Drop IoT Project Builder

**Cayenne**

[About Cayenne](#)

[Documentation](#)

## General settings

Webhook ID \*

cayenne

Webhook format \*

JSON

## Endpoint settings

Base URL \*

https://loramyldevices.com/v1/networks/ttn

Downlink API key

.....



The API key will be provided to the endpoint using the "X-Downlink-APIkey" header

# Hands-on, 15': Dashboard as a service

Choose a dashboard service\* and a transport protocol.

Check the API docs to understand the payload format.

Send data "as a device" with [curl](#) or with the [mqtt](#) CLI.

The CLI runs on the Raspberry Pi or on your laptop.

\*Try ThingSpeak, Cayenne or [ThingsBoard.io](#)

# Glue code

**Glue code** is a simple way to integrate service APIs.

A custom adapter acts as a client of both services, e.g. to get data from a LoRaWAN backend to a dashboard.

It converts payload formats, can be hosted anywhere.



# Node.js glue code

`.js`<sup>MQTT</sup>, `.js`<sup>HTTP</sup>

...

```
var client = new ttn.Client('eu', appId, key);
client.on('message', (devId, msg) => {
  var bytes = msg.payload_raw;
  var x = ((bytes[0] << 8) | bytes[1]) / 100.0;
  http.post('http://api.thingspeak.com/update?'
    + 'api_key=' + writeApiKeys[msg.dev_id]
    + '&field1=' + x);
});
```

# Serverless lambda functions

**Serverless** lambda functions are cloud hosted handlers.

Conceptually, a server is started for each Web request.

No resources are consumed between Web requests.

This execution model works well for glue code.

E.g. **AWS Lambda**, **Azure Functions** or **Vercel**.

# Serverless Node.js with Vercel

**Vercel** provides hosting for serverless functions.

On you MacOS, Windows or Linux computer:

- Install *vercel* with `$ npm install -g vercel`
- Get examples `$ git clone https://github.com/vercel/vercel` then `$ cd /examples`
- Or use examples linked from the following pages
- Deploy with `$ vercel`

# Serverless Node.js glue code

.js

Creating a Web service in Node.js ...

```
let server = http.createServer((req, res) => {  
  res.end("200 OK");  
});
```

... becomes exporting a handler in Vercel Node.js:

```
module.exports = (req, res) => {  
  res.end("200 OK");  
};
```

# Hands-on, 15': Glue code

Configure the **TTN to ThingSpeak adapter** glue code.

Host the code serverless or on your laptop via a relay.

Use **curl** to simulate calls from the TTN backend:

```
$ curl -v http://HOST_OR_IP/ --data  
'{"end_device_ids": {"device_id": "fhnw-iot-device-1"},  
"uplink_message": {"frm_payload": "CPwQaA=="}}'
```



# Self-hosted dashboard

A self-hosted dashboard backend includes:

- A way to run services (and keep them running)
- A service with an API to store or cache data
- A service serving dashboard resources

The backend can be hosted locally or in the cloud, storage or cache can be a database, broker or both.

# Docker

**Docker** provides OS-level virtualisation/containers.

Use it to run services on **Windows**, **MacOS** and **Linux**.

Docker runs on a Pi Zero, but most containers do not\*.

\*ARMv6 instructions are usually not supported.

# InfluxDB

**InfluxDB** is an open source time-series database.

- To run **InfluxDB on Docker**, type:  

```
$ docker run --name influxdb -p 8086:8086 \
influxdb
```
- To set it up, open **the Web UI** or a new terminal:  

```
$ docker exec -it influxdb /bin/bash
$ influx setup
```

# Getting data into InfluxDB

InfluxDB has a number of mechanisms to get data in:

- [Telegraf](#), a data collection agent, supports [MQTT](#).
- [Data formats](#) like [Line Protocol](#), [CSV](#) and [JSON](#).
- [InfluxDB scrapers](#) can collect data from any HTTP endpoint using the [Prometheus data format](#).
- InfluxDB [integrations](#) with systems like [Kafka](#).

# Telegraf

**Telegraf** is a data collection agent with many plugins.

- To run **Telegraf on Docker**, type:  

```
$ docker run --net=container:influxdb  
telegraf
```
- To enable the MQTT plugin, use a *telegraf.conf*:  

```
$ docker exec -it telegraf /bin/bash  
$ nano telegraf.conf #see next page
```

# Telegraf MQTT input

.conf

The *telegraf.conf* for an MQTT to InfluxDB adapter:

```
[[inputs.mqtt_consumer]]
  servers = ["ssl://MQTT_HOST_OR_IP:8883"]
  topics = ["TOPIC/SUBTOPIC"]
  data_format = "json" ...

[[outputs.influxdb]]
  urls = ["http://INFLUXDB_HOST_OR_IP:8086"]
  database = "telegraf"
```

# Grafana

**Grafana** is an open source Web dashboard backend.

It integrates nicely with InfluxDB and other sources.

- To run **Grafana on Docker**, type:  
`$ docker run -d -p 3000:3000 grafana/grafana`
- Then configure it to connect to InfluxDB:  
`http://127.0.0.1:3000/`
- And create some graph views to display data.

LoRa WAN demo project by



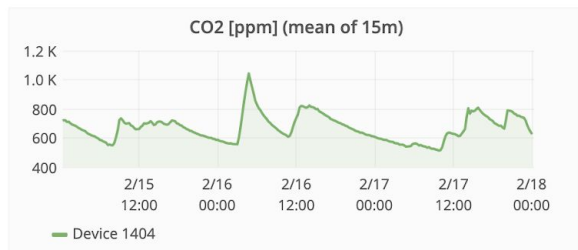
[CO2, temperature, and humidity sensor](#)



Node **1404** (CO2)



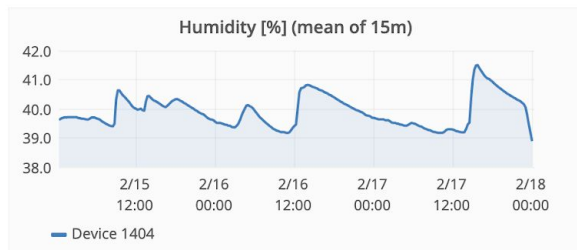
LoRa infrastructure provided by



Current

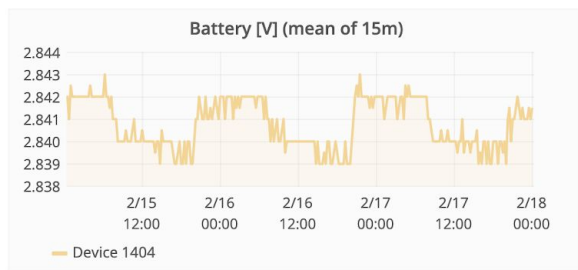
**625.5**

**ppm**



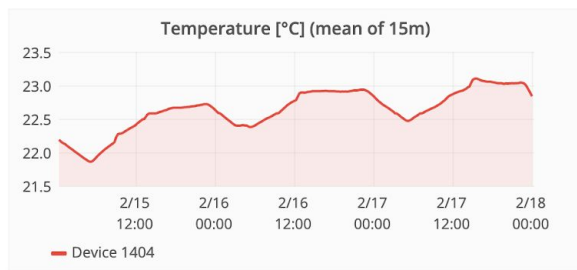
Current

**38.9 %**



Current

**2.84 V**



Current

**22.84 °C**



# Hands-on, 15': Docker hosted dashboard

Install Docker on your computer (not Raspberry Pi).

Run InfluxDB and run/create a Grafana dashboard.

Run Telegraf to get data from [test.mosquitto.org](https://test.mosquitto.org).

Send data "as a device" with [mqtt](#), to Mosquitto.

How does the reference model of this setup look?

# Graph libraries

For custom dashboards, graph libraries are available:

- [Plotly.js](#) is quite easy to [get started](#) in Node.js
- [CanvasJS](#) has a big collection of [React Charts](#)
- [Google Charts](#) has been around for a while

There are many other libraries, make sure to check (long term) availability and source code license.

# Summary

We created Web dashboards & apps for sensor data.

We saw how data gets to a hosted dashboard service.

We wrote adapter glue code for backend integration.

We set up a self hosted dashboard with Docker.

Next: Rule Based Integration of IoT Devices.

# Feedback or questions?

Write me on Teams or email

[thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

Thanks for your time.