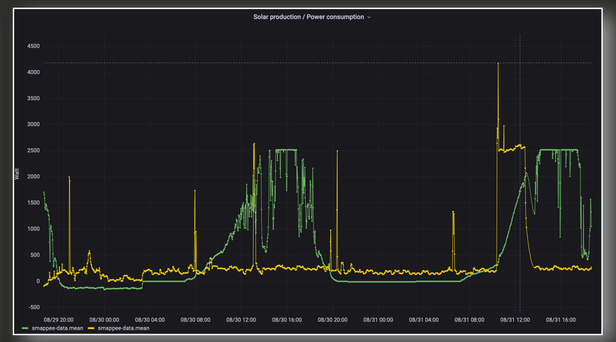
30/09/2022— A LUNATECH PRESENTATION

Setting up a home energy monitoring system on Raspberry Pi with OSS

LUNACONF



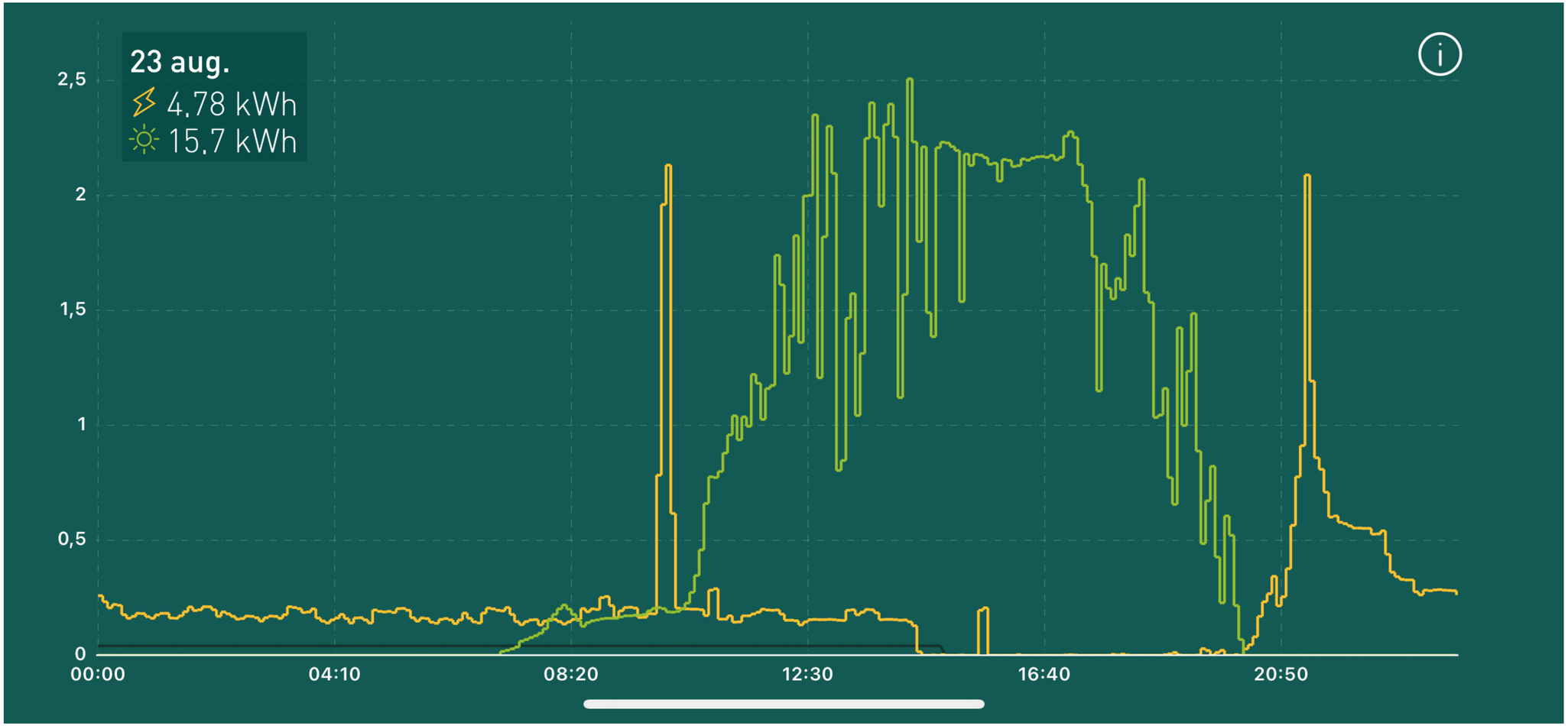
Eric Loots - Lunatech

Agenda

- Why did I start this project?
- Goals of the project
- Choosing the core OSS components
- Practical experience while implementing the project
- Conclusions
- What’s next

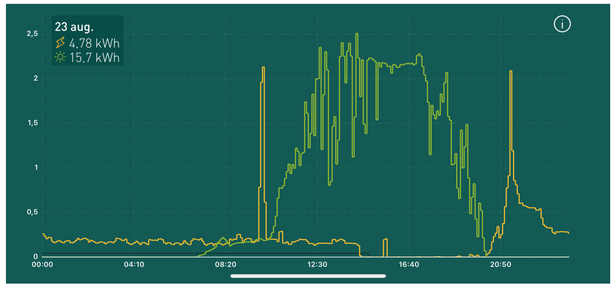
The why…

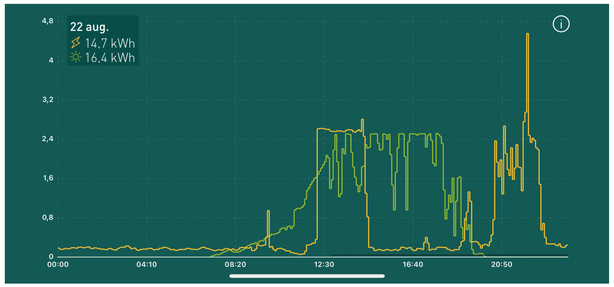
- Smappee keeps historical data for some time.
- Finest granularity are 5 minute averages:

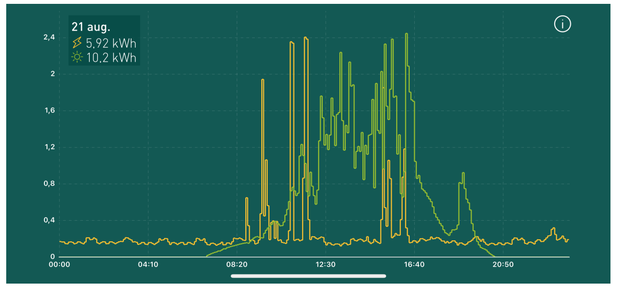


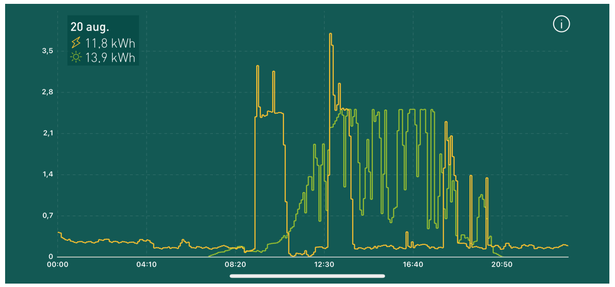
The why…

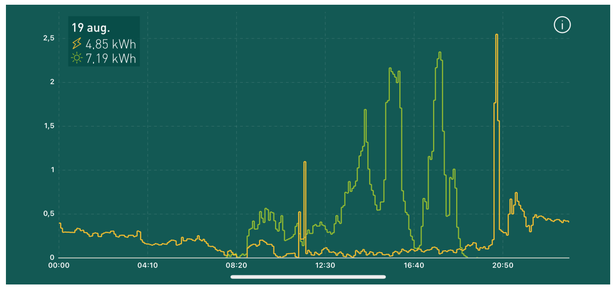
The why…

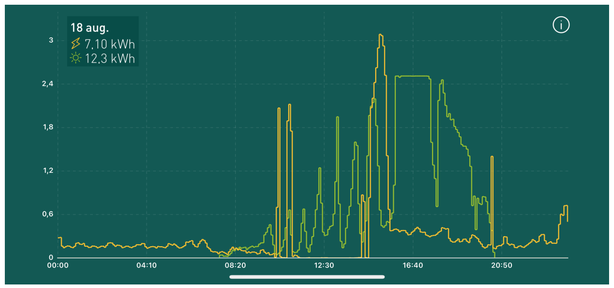

23 aug.
4,78 kWh
15,7 kWh

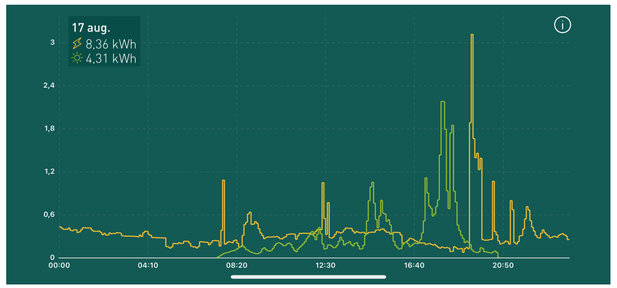

22 aug.
14,7 kWh
16,4 kWh

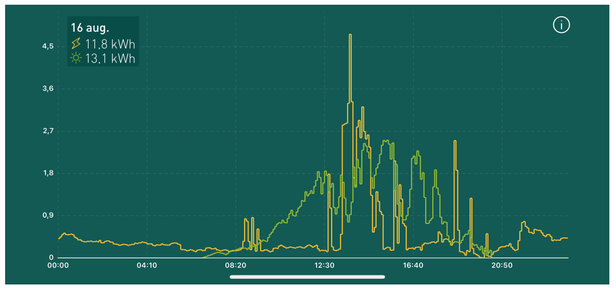

21 aug.
5,92 kWh
10,2 kWh

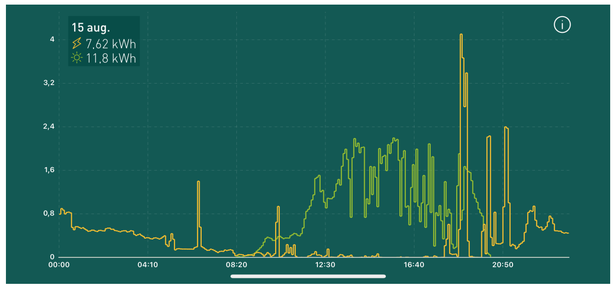

20 aug.
11,8 kWh
13,9 kWh

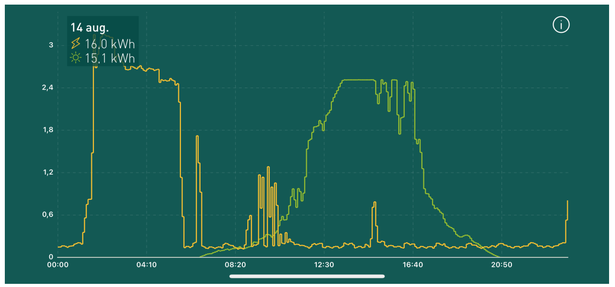

19 aug.
4,85 kWh
7,19 kWh

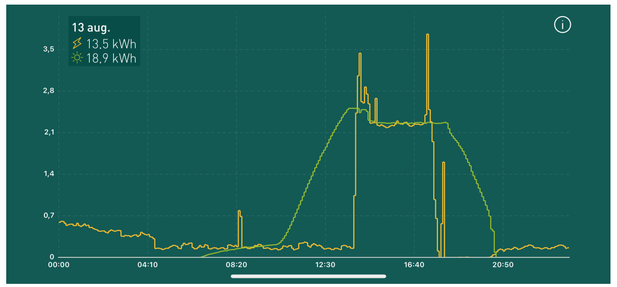

18 aug.
7,10 kWh
12,3 kWh

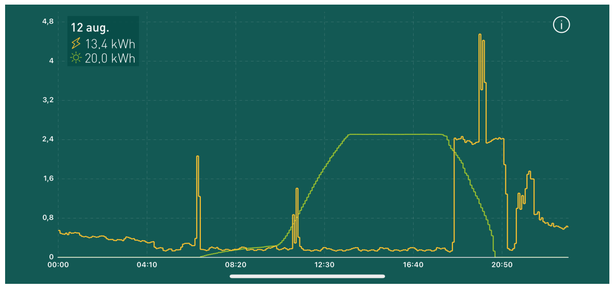

17 aug.
8,36 kWh
4,31 kWh

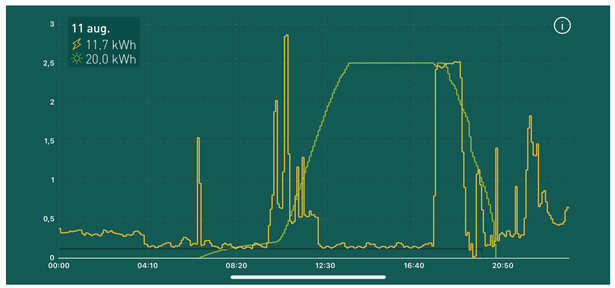

16 aug.
11,8 kWh
13,1 kWh

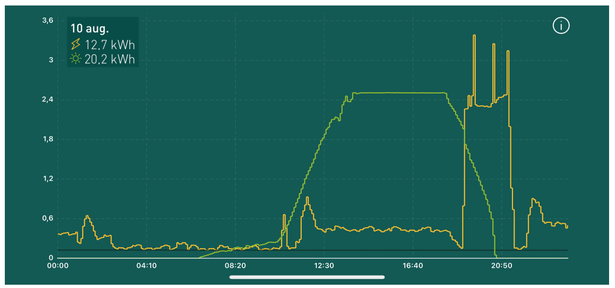

15 aug.
7,62 kWh
11,8 kWh

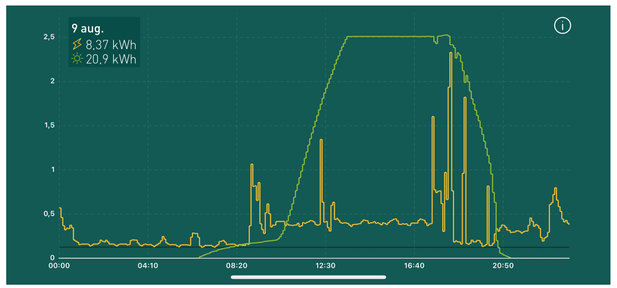

14 aug.
16,0 kWh
15,1 kWh

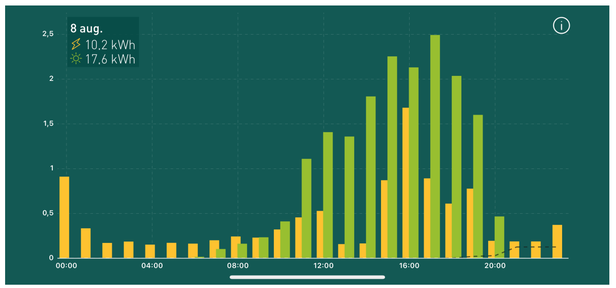

13 aug.
13,5 kWh
18,9 kWh


12 aug.
13,4 kWh
20,0 kWh

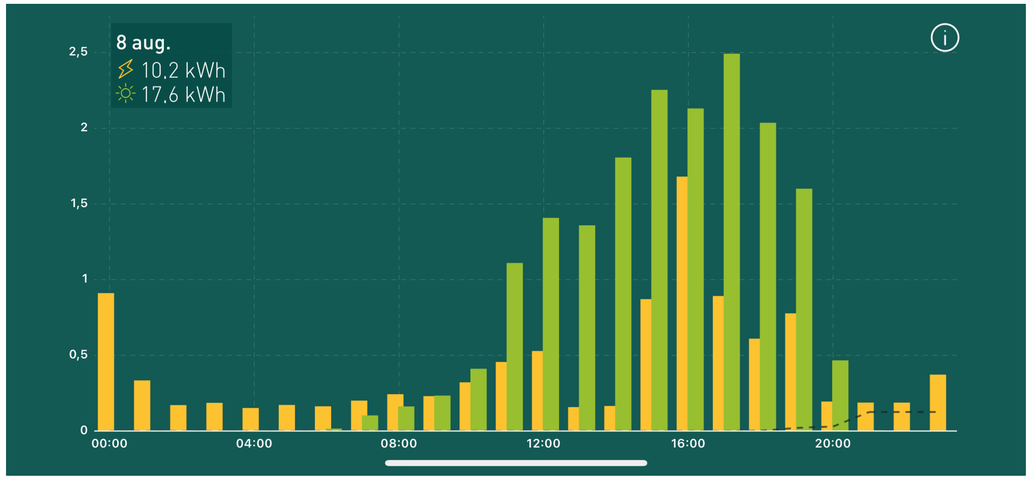

11 aug.
11,7 kWh
20,0 kWh


10 aug.
12,7 kWh
20,2 kWh

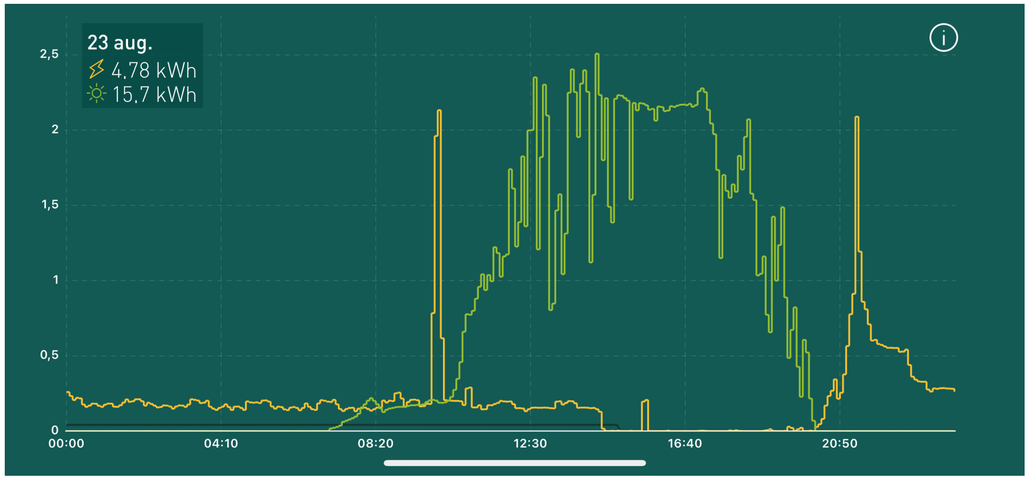

9 aug.
8,37 kWh
20,9 kWh


8 aug.
10,2 kWh
17,6 kWh

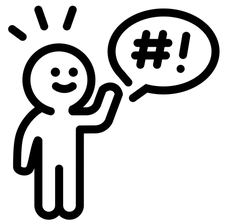
The why…



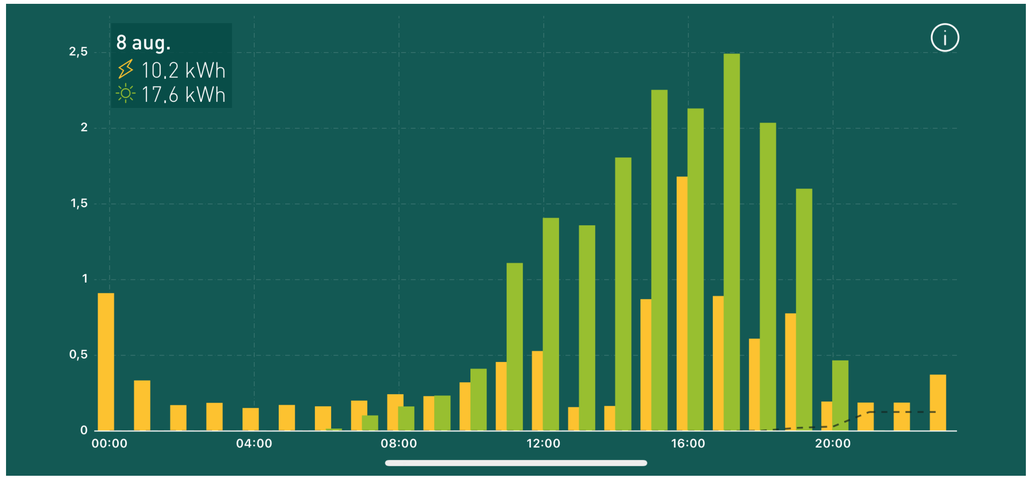
± 2 weeks



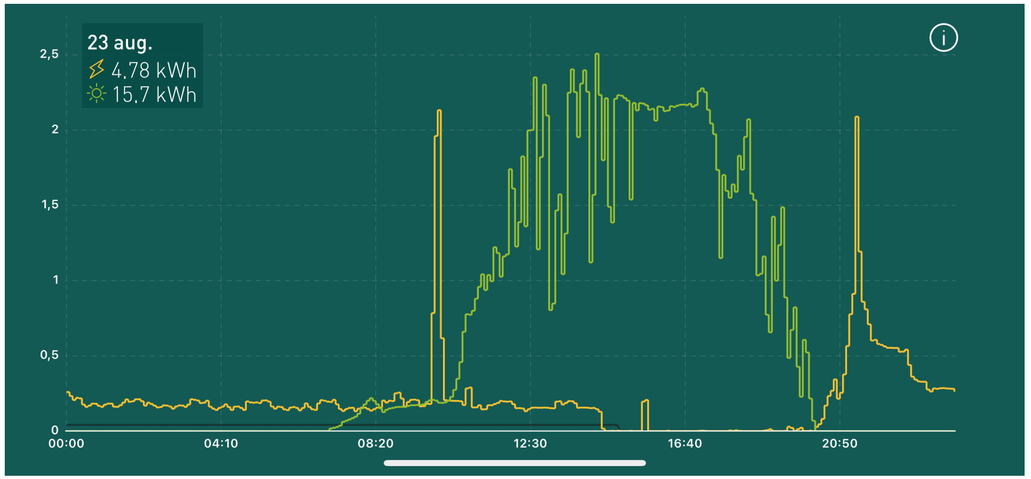
After ± two weeks: 1 hour averages instead of 5 minute averages !



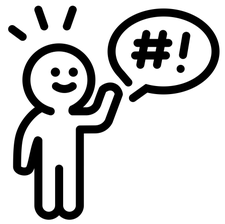
The why…



± 2 weeks



After ± two weeks: 1 hour averages instead of 5 minute averages !



The why…

What shall we do about this ?

- Complain to Smappee ?
- Roll our own ?

The why…

What shall we do about this ?

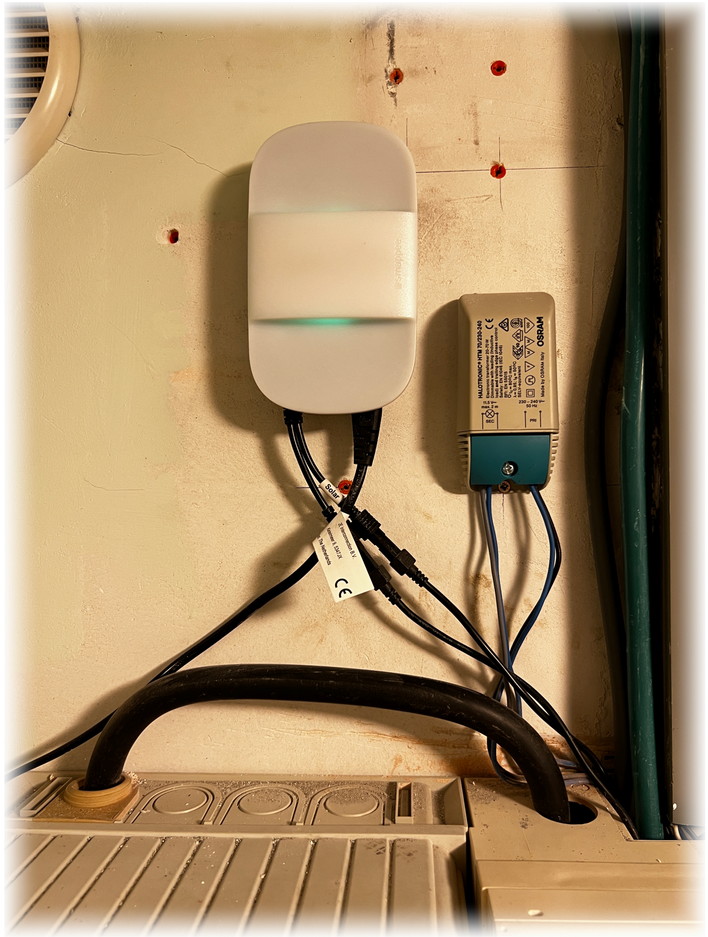
- Complain to Smappee ?
- Roll our own ?

Let’s explore rolling our own !

Goals of the project

- Run all the bits on a Raspberry Pi with an USB-attached SSD
- Implement the monitoring system that
  - avoids programming if possible
  - uses *Open Source Software* only
  - captures all knowledge as to make this easily reproducible [for others]
  - automates the installation of the OS and required software

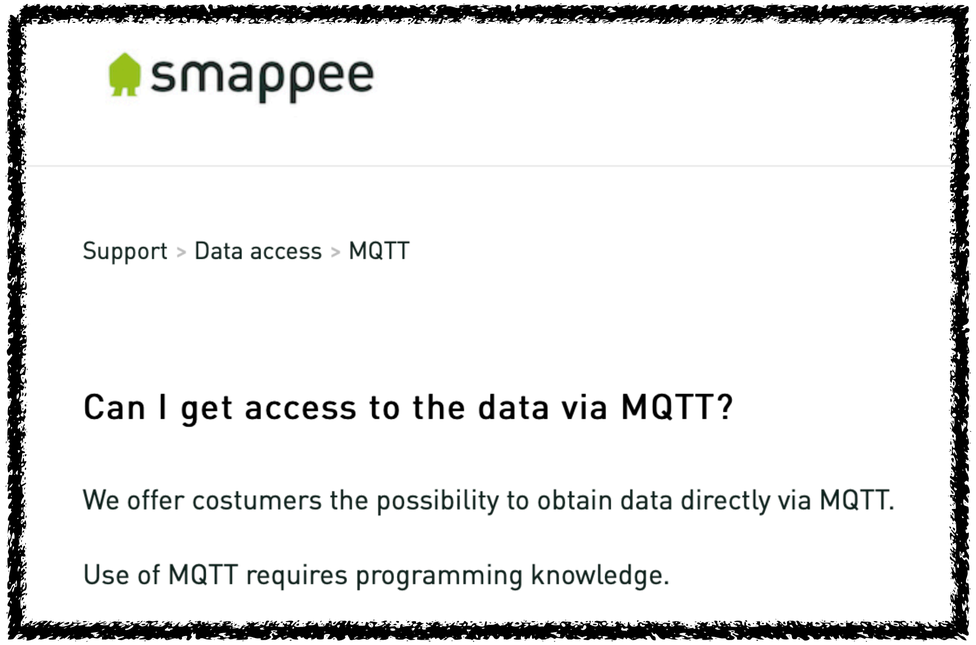
The Smappee device



- 4-channel power measurement
- 1 second sample interval
- MQTT client

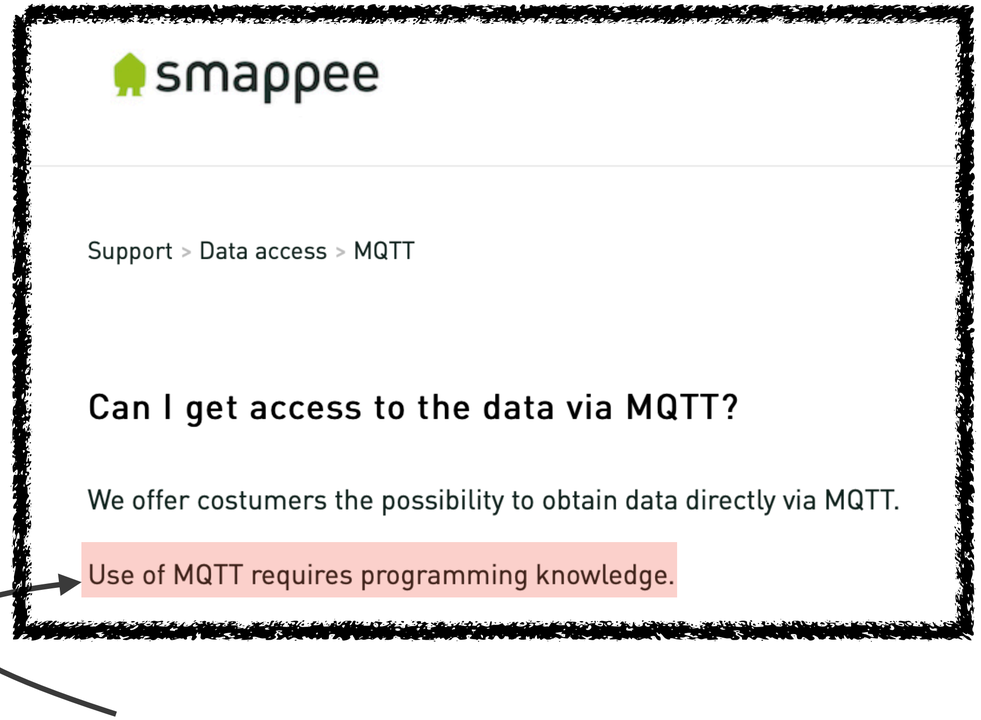
What are our options?

- Smappee offers access to data in two ways
  - via REST API (paying service)
  - via MQTT:



What are our options?

- Smappee offers access to data in two ways
  - via REST API (paying service)
  - via MQTT:

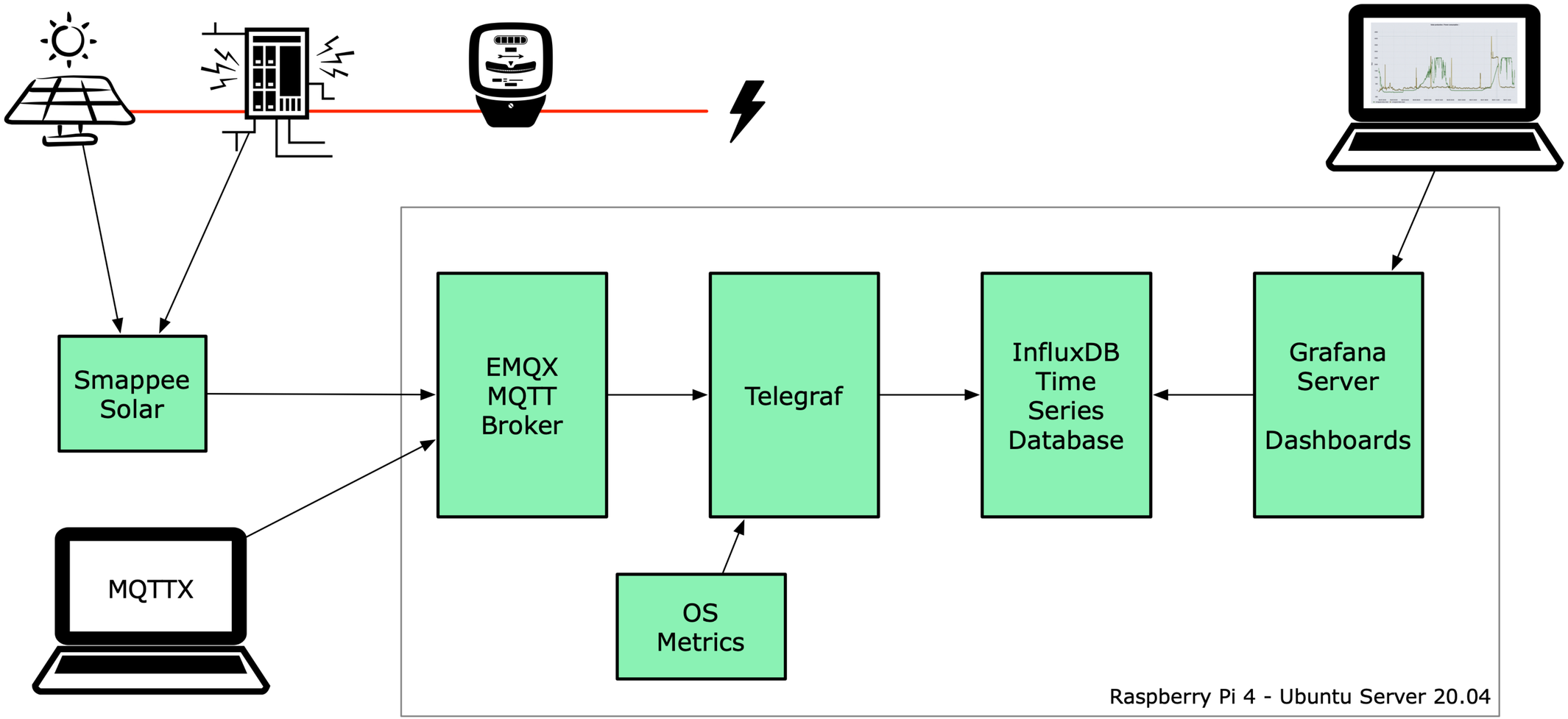


Sounds interesting, but do I really need to program?

The base platform

- Hardware:
  - Raspberry Pi 4 Model B - 8GB (smaller will do)
  - Storage: Samsung 250GB SSD attached to a USB-3 port
- OS:
  - 64-bit Ubuntu Server 20.04.4 LTS
    - Has cloud-init (version 21.4) support
    - Can boot of a USB disk with a gotcha

Home monitoring system setup



The software components

Component	Description	Version	OS support
EMQX	MQTT broker	5.0.4	Ubuntu 20.04
MQTTX	Cross-platform MQTT 5.0 client tool, GUI & CLI	1.6.0	*nix, Mac, Windows
InfluxDB	Time series database	1.8.10	≥ Ubuntu 20.04
Telegraf	An agent for collecting, processing, aggregating, and writing metrics	1.23.3	≥ Ubuntu 20.04
Grafana Server	Query, visualize, alert on and understand your metrics	9.0.6	

EMQX forces the utilisation of Ubuntu 20.04

Configuring/testing MQTT

- We configure Smappee for our MQTT Broker (EMQX)

  our broker lives at 192.168.68.201:1883

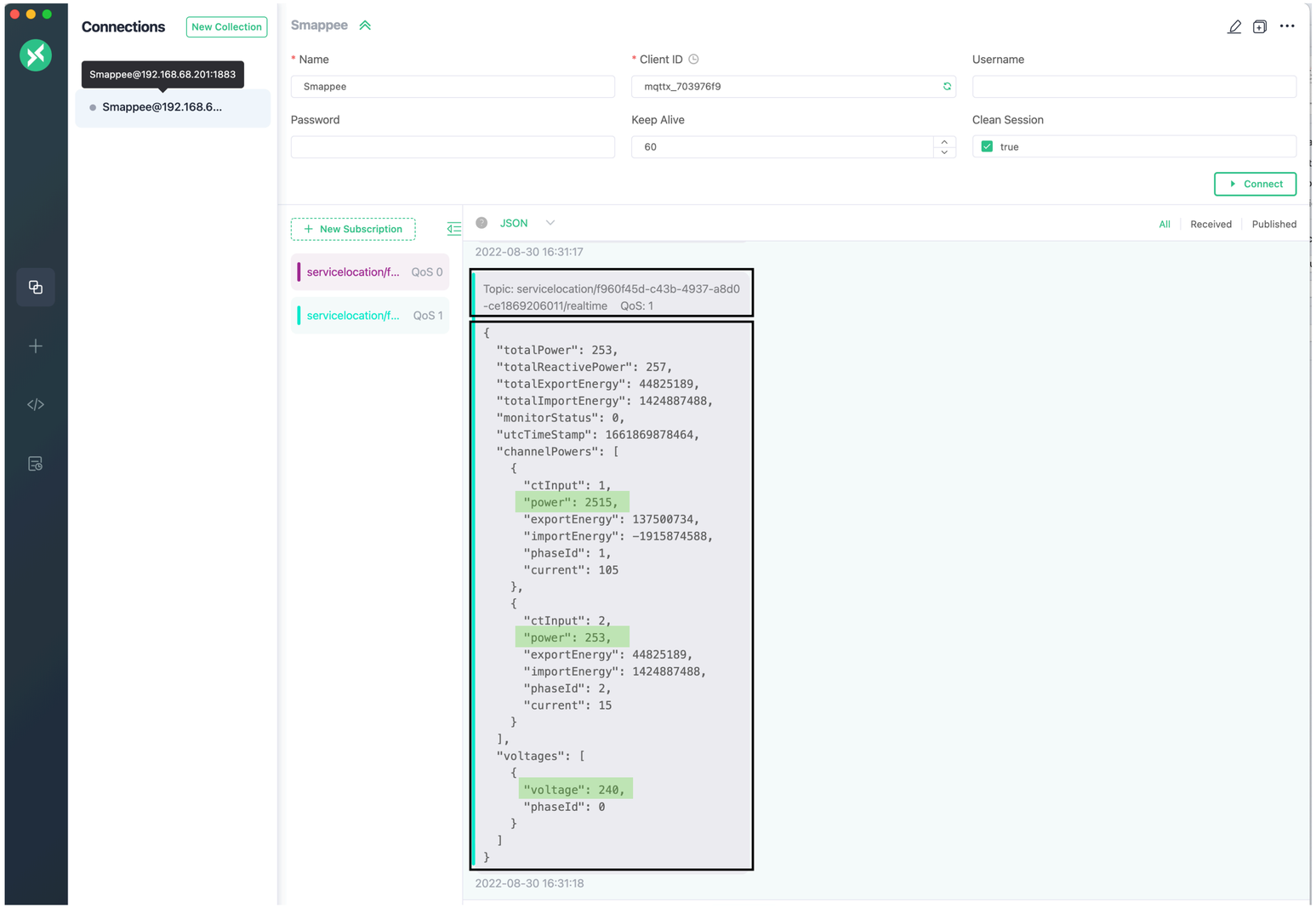
Advanced config parameters	Setting
Webportal password	admin
Active power lower limit (color green->yellow)	0
Active power upper limit (color yellow->orange)	0
MDNS hostname	Smappee11070075
MQTT local broker (e.g. tcp://192.168.0.48:1883)	tcp://192.168.68.20
MQTT local broker username	
MQTT local broker password	
Switch 5 minute read delay	0

Apply changes and restart monitor

Advanced action	
Restart monitor	apply
Clear all data (historical data + appliances data) and restart monitor	apply
Clear only appliances data and restart monitor	apply
Reset active power peak values	apply
Reset sensor cache	apply
Autocommissioning validation	stop start reset values
Enable/disable green breathing	☑
Enable/disable command control look ahead schedule recover	☑
Enable/disable MAC address arp-scan	☐
Reset MAC address scanning cache	apply
Enable/disable MDNS (monitor will reboot)	☑
Advanced	☑

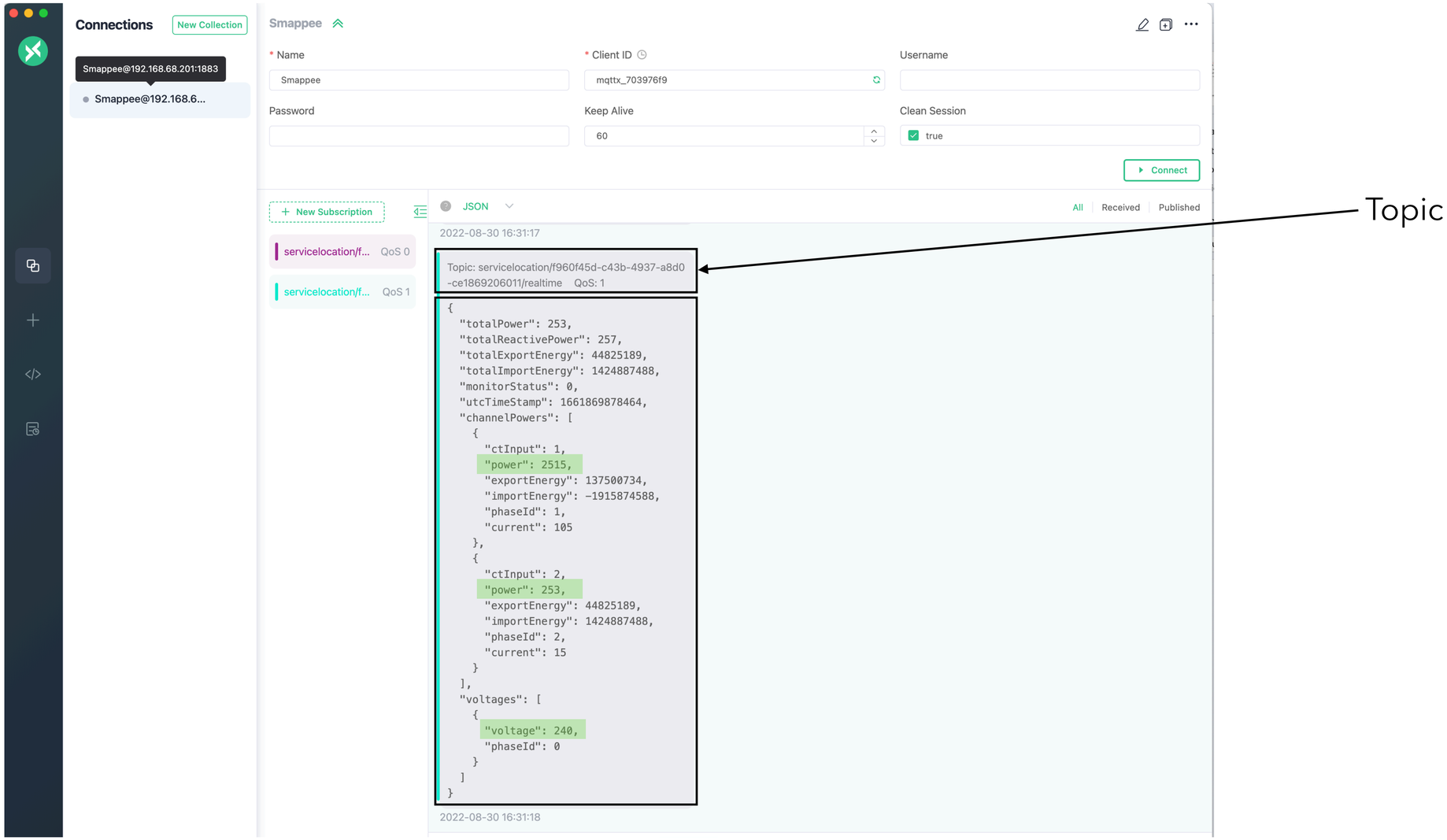
Configuring/testing MQTT

- Let's use MQTTX to find out what data our Smappee sends



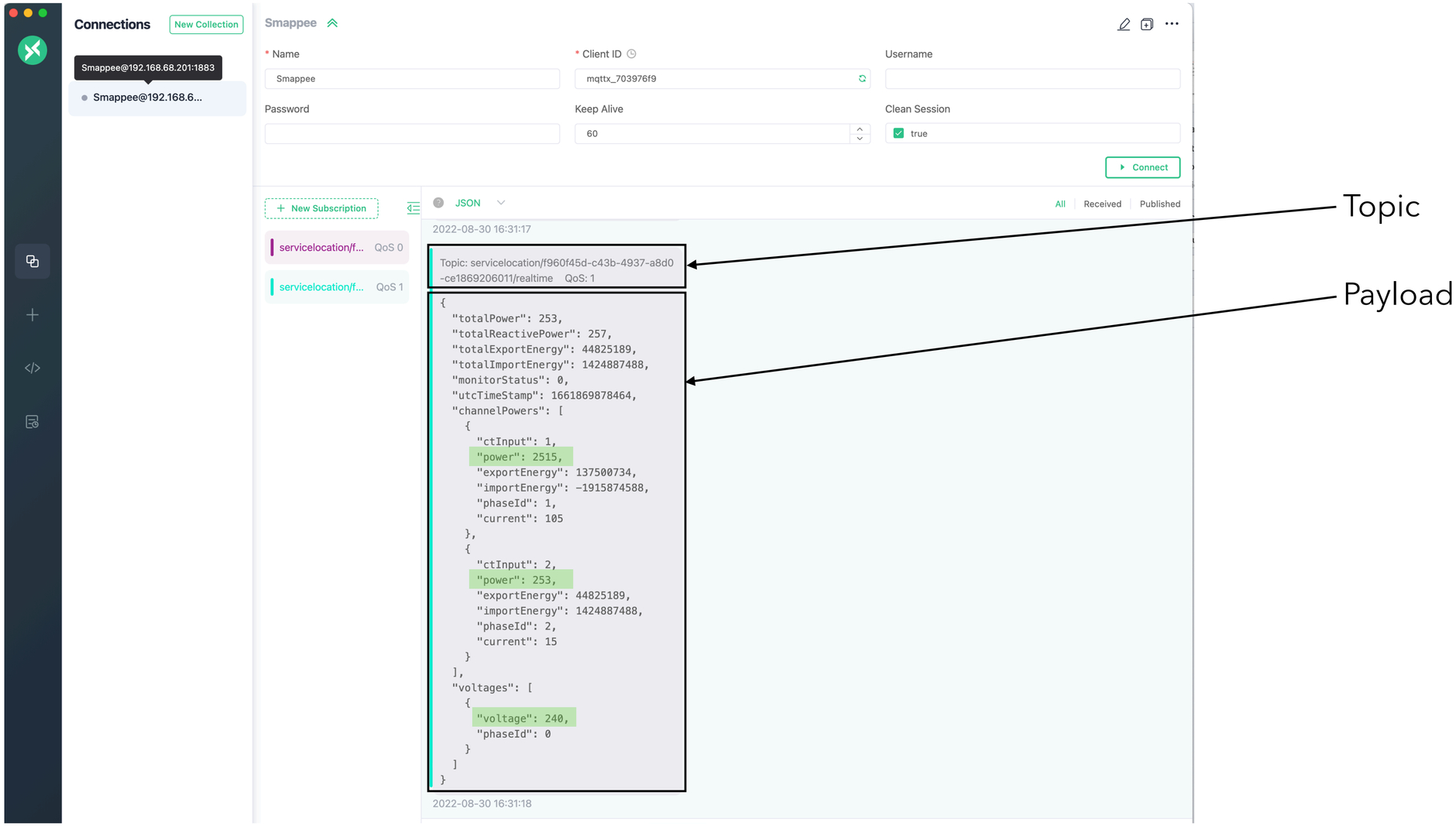
Configuring/testing MQTT

- Let's use MQTTX to find out what data our Smappee sends



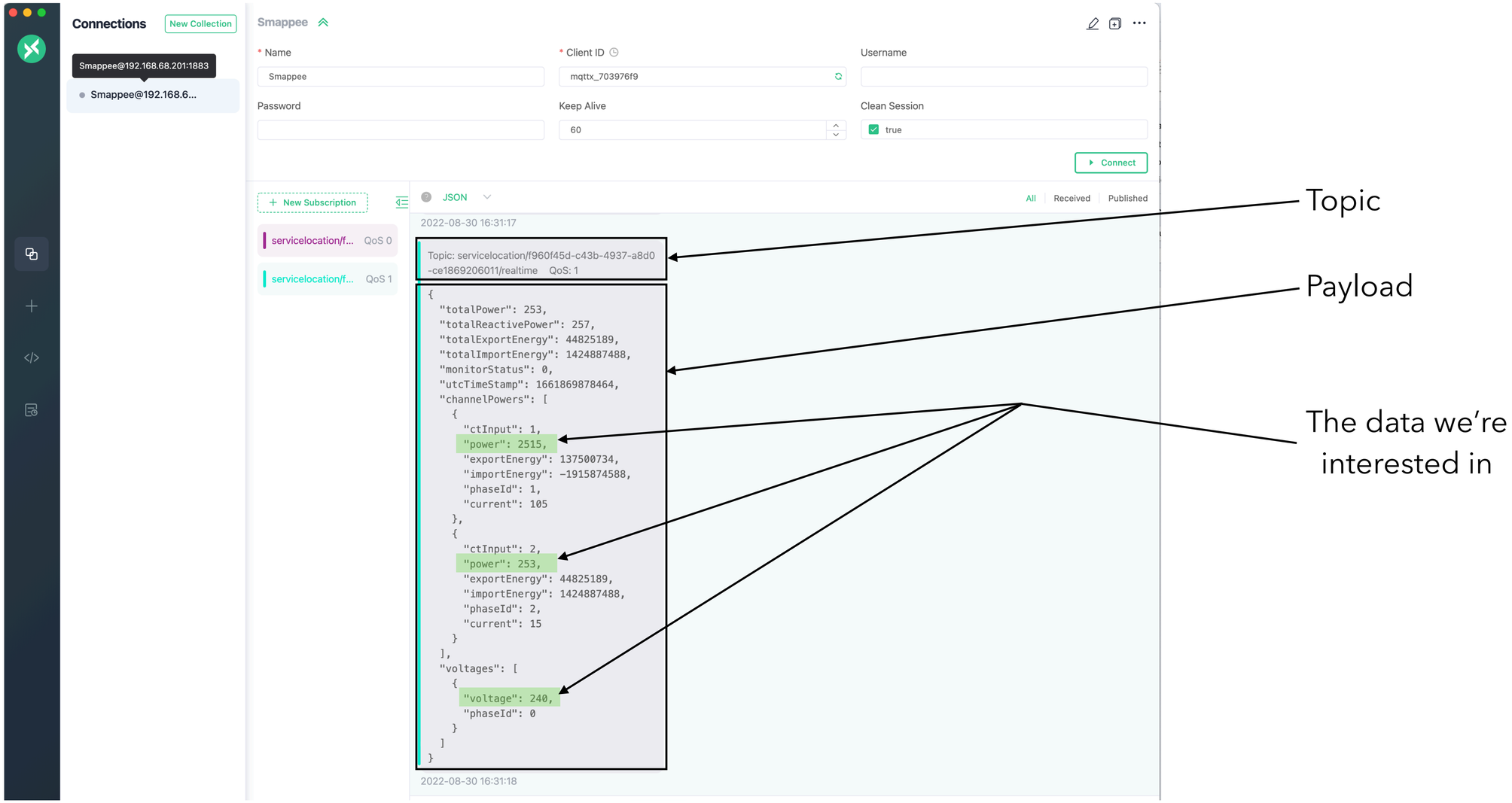
Configuring/testing MQTT

- Let's use MQTTX to find out what data our Smappee sends



Configuring/testing MQTT

- Let's use MQTTX to find out what data our Smappee sends



Telegraf configuration

- Telegraf allows to select and map data fields, aggregate them, and write them out
- This is done via configuration of *Telegraf plugins*

`servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime`


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration

- We will configure these plugins:
  - *mqtt_consumer* input
  - *json_v2* parser
  - *regex* processor
  - *basicstats* aggregator
  - *influxdb* output
- Let's build the configuration one step at a time

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - mqtt_consumer input

- Telegraf plugins share a common configuration space: be precise!
- Use *name_override* and *name_pass* to wire things together
- We start with the mqtt_consumer input plugin:

```
[[inputs.mqtt_consumer]]
  alias = "smappee-test"
  name_override = "smappee-data-test"
  servers = ["tcp://192.168.68.201:1883"]
  topics = [
    "servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime"
  ]
  # The "host" tag is irrelevant in this use case. Drop it
  tagexclude = ["host"]
  data_format = "json_v2"
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Field selection/naming

- Next we use the json_v2 parser
- Use the *gjson.dev* playground to your advantage

```
[[inputs.mqtt_consumer]]
  alias = "smappee-test"
  name_override = "smappee-data-test"
  servers = ["tcp://192.168.68.201:1883"]
  topics = [
    "servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime"
  ]
  # The "host" tag is irrelevant in this use case. Drop it
  tagexclude = ["host"]
  data_format = "json_v2"
  [[inputs.mqtt_consumer.json_v2]]
    [[inputs.mqtt_consumer.json_v2.field]]
      path = "channelPowers.#(ctInput==2).power"
      rename = "consumption"
      type = "float"
    [[inputs.mqtt_consumer.json_v2.field]]
      path = "channelPowers.#(ctInput==1).power"
      rename = "solar"
      type = "float"
    [[inputs.mqtt_consumer.json_v2.field]]
      path = "voltages.#.voltage"
      type = "float"
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Topic remapping

- The topic is a bit clunky for our purposes
- Let's rename using the *regex* processor
  - Note the utilisation of the *name_pass* directive

```
[[processors.regex]]
  namepass = ["smappee-data-test"]
  [[processors.regex.tags]]
    key = "topic"
    pattern = ".*/(.*)/.*"
    replacement = "smappee/${1}"
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Topic remapping

- The topic is a bit clunky for our purposes
- Let's rename using the *regex* processor
  - Note the utilisation of the *name_pass* directive

```
[[processors.regex]]
  namepass = ["smappee-data-test"]
  [[processors.regex.tags]]
    key = "topic"
    pattern = ".*/(.*)/.*"
    replacement = "smappee/${1}"
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime

smappee/f960f45d-c43b-4937-a8d0-ce1869206011

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Aggregation

- Data arriving at 1 sample/s
- Aggregate to 1 minute average
  - Drop the original data

```
[[aggregators.basicstats]]
  namepass = ["smappee-data-test"]
  ## The period on which to flush & clear the aggregator.
  period = "60s"

  ## If true, the original metric will be dropped by the
  ## aggregator and will not get sent to the output plugins.
  drop_original = true

  ## Configures which basic stats to push as fields
  stats = ["mean"]
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Output to InfluxDB

- Persisting data to InfluxDB is a breeze

```
[[outputs.influxdb]]
  namepass = ["smappee-data-test"]
  alias = "smappee-out"
  urls = ["http://192.168.68.201:8086"]
  database = "smappee_test"
  username = "demo"
  password = "demo"
```

servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime


```
{
  "totalPower": 253,
  "totalReactivePower": 257,
  "totalExportEnergy": 44825189,
  "totalImportEnergy": 1424887488,
  "monitorStatus": 0,
  "utcTimeStamp": 1661869878464,
  "channelPowers": [
    {
      "ctInput": 1,
      "power": 2515,
      "exportEnergy": 137500734,
      "importEnergy": -1915874588,
      "phaseId": 1,
      "current": 105
    },
    {
      "ctInput": 2,
      "power": 253,
      "exportEnergy": 44825189,
      "importEnergy": 1424887488,
      "phaseId": 2,
      "current": 15
    }
  ],
  "voltages": [
    {
      "voltage": 240,
      "phaseId": 0
    }
  ]
}
```

Telegraf configuration - Summary

- The complete configuration at a glance

```
[[inputs.mqtt_consumer]]
  alias = "smappee-test"
  name_override = "smappee-data-test"
  servers = ["tcp://192.168.68.201:1883"]
  topics = [
    "servicelocation/f960f45d-c43b-4937-a8d0-ce1869206011/realtime"
  ]

  tagexclude = ["host"]
  data_format = "json_v2"

  [[inputs.mqtt_consumer.json_v2]]

    [[inputs.mqtt_consumer.json_v2.field]]
      path = "channelPowers.#(ctInput==2).power"
      rename = "consumption"
      type = "float"

    [[inputs.mqtt_consumer.json_v2.field]]
      path = "channelPowers.#(ctInput==1).power"
      rename = "solar"
      type = "float"

    [[inputs.mqtt_consumer.json_v2.field]]
      path = "voltages.#.voltage"
      type = "float"
```

```
[[processors.regex]]
  namepass = ["smappee-data-test"]
  [[processors.regex.tags]]
    key = "topic"
    pattern = ".*/(.*)/.*"
    replacement = "smappee/${1}"


[[aggregators.basicstats]]
  namepass = ["smappee-data-test"]
  ## The period on which to flush & clear the aggregator.
  period = "60s"
  drop_original = true
  stats = ["mean"]


[[outputs.influxdb]]
  namepass = ["smappee-data-test"]
  alias = "smappee-out"
  urls = ["http://192.168.68.201:8086"]
  database = "smappee_test"
  username = "demo"
  password = "demo"
```

InfluxDB configuration

- Using InfluxDB version 1
- The configuration is extremely simple using the InfluxDB CLI:
  - Create the database(s)
  - Create a user with write privileges

```
akkapi@node-1:~$ influx
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
> create database smappee_test
> create user demo with password 'demo' with all privileges;
```

  - …

InfluxDB version 1 versus version 2

- InfluxDB version 2 …
  - **TICK** stack in a single binary:
    - **T**elegraf
    - **I**nfluxDB
    - **C**hronograf (Dashboards, Admin, Alerting)
    - **K**apacitor (Real-time streaming data processing engine)
  - Has much better security than version 1, authentication baked in everywhere
  - Has Flux Query (and still supports InfluxQL)
  - InfluxDB tasks for downsampling data
  - Has a great upgrade story as I experienced myself
  - …

Ubuntu - takeaways

- Using cloud-init, a system can be provisioned from scratch in about 20 minutes
- Ubuntu 20.04 is not the most recent version
  - Cannot directly boot of an external SSD. An SD still has to be present

```
akkapi@node-1:~$ mount -t ext4,vfat
/dev/sda2 on / type ext4 (rw,relatime)
/dev/sda1 on /boot/firmware type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=ascii,shortname=mixed,errors=remount-ro)
akkapi@node-1:~$ _
```

  - Flashing an SD takes ±5 minutes
  - Flashing an SSD takes ±18 seconds

cloud-init - takeaways

- cloud-init is a method for cross-platform instance initialisation
- Can be used for bare-metal installations
- Performs user creation, execute custom scripts, install packages, create files, …
- Has rather poor docs, but this has improved *a lot* recently
- Read the Module Reference
- Supports jinja template rendering:
  - cloud-init *meta-data* file contains key/value pairs
  - keys used in user-data will be dereferenced by jinja

```
instance_id: cloud-image
influx_db1: smappee_test
influx_user: demo
influx_passwd: demo
```

meta-data

```
runcmd:
  - |
    sudo -u akkapi influx <<ENDCFGINFLUX
    create database {{ ds.meta_data.influx_db1 }};
    create user {{ ds.meta_data.influx_user }} with password '{{ ds.meta_data.influx_passwd }}' with all privileges;
    ENDCFGINFLUX
```

user-data

cloud-init - jinja

- Your meta-data key-value pairs may not be where you expect them to be

```
akkapi@node-4:~$ cloud-init query --all
{
 "_beta_keys": [
  "subplatform"
 ],
 "availability_zone": null,
 "base64_encoded_keys": [],
 "cloud_name": "unknown",
 "distro": "ubuntu",
 "distro_release": "focal",
 "distro_version": "20.04",
 "ds": {
  "_doc": "EXPERIMENTAL: The structure and format of content scoped under the 'ds' key may change in subsequent releases of cloud-init.",
  "meta_data": {
   "dsmode": "net",
   "influx_db1": "smappee_test",
   "influx_passwd": "demo",
   "influx_user": "demo",
   "instance_id": "cloud-image"
  }
 },
 "instance_id": "nocloud",

<elided>
```

cloud-init - jinja

- Flashing

```
$ flash -n node-1 -j -m cloud-init/meta-data -u cloud-init/akka-pi-os-dhcp-64-ssd.yml \
   https://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04.5-preinstalled-server-arm64+raspi.img.xz
```

What's next?

- Explore InfluxDB version 2
- Add a new Smappee device with 12 channels to my home setup
- More Grafana dashboards
- More aggregations
- Alerts
- Automatic equipment control based on data (for example, an electrical water boiler)

Conclusions

- Learned a ton about InfluxDB, Telegraf, EMQX, Grafana
- Documentation is a weak point of many of the OSS projects, but things are improving
- Implemented the monitoring system that:
  - ✓ avoided programming
  - ✓ uses *Open Source Software* only
  - ✓ captures all knowledge through installation and configuration automation
  - ✓ automates the installation of the OS and required software

Setting up a home energy monitoring system on Raspberry Pi with OSS

Questions?