

Rapport de projet – Profilage d'appareils électriques avec une pince ampérométrique et une carte ESP8266

Sommaire

Introduction.....	2
1. Configuration capteur / carte.....	2
1.1. ESP8266.....	2
1.1.1. Features.....	3
1.1.2. SDK.....	4
1.1.3. Pins description.....	4
1.2. Capteur de courant SCT013.....	5
1.3. Horloge Tiny RTC.....	7
1.4. Réalisation du montage.....	7
1.5. Téléversement du code dans l'ESP8266.....	8
2. Configuration serveur.....	9
2.1. Principe.....	9
2.2. Configuration générale de Docker.....	9
2.3. Configuration de la base de données InfluxDB.....	10
2.4. Configuration de node-red.....	10
2.5. Configuration de Grafana.....	12
3. Profils de consommation observés.....	14
3.1. Profil d'une consommation d'un sèche-cheveux.....	14
3.2. Profil de consommation d'un fer à repasser.....	15

Introduction

Notre projet consiste à mettre en place un dispositif capable d'enregistrer de manière persistante dans le cloud des profils de consommation de courant d'appareils électriques. Nous disposons pour cela d'une pince ampérométrique qui une fois clipsée sur un câble d'alimentation permet d'effectuer au cours du temps des mesures de l'intensité circulant dans le câble de manière non intrusive.

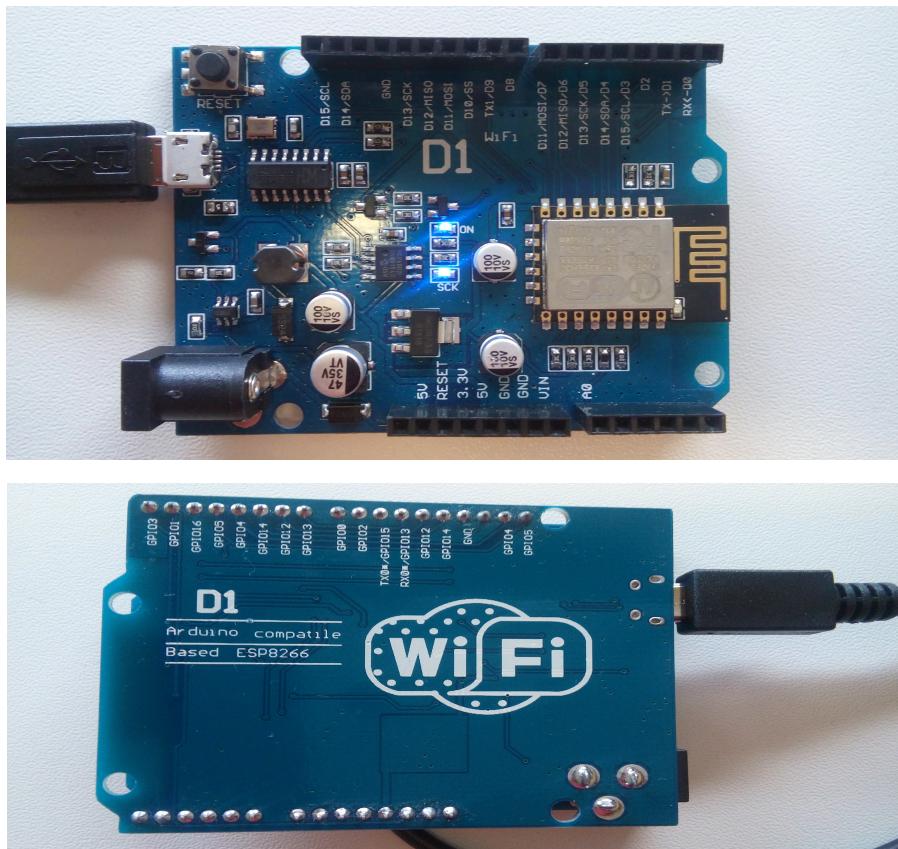
Ces relevées de mesures sont directement transmis par udp via une carte wifi ESP8266 à un serveur de collecte backend qui les enregistre dans une base de données temporelles. Un outil de visualisation permet ensuite de consulter sous forme de tableau de bord les différents profils collectés en les différenciant par id de capteurs et région.

1. Configuration capteur / carte

Pour cette expérience nous avons utilisé:

1. Un ESP8266
2. Un capteur de courant SCT013
3. Prototype shield v.5
4. Une horloge externe Tiny RTC I2C modules

1.1. ESP8266



1.1.1. Features

- **MCU Tensilica Xtensa LX106**
 - 32 bits RISC
 - clock: 80 Mhz (can be doubled to 160 Mhz)
- **RAM**
 - Data: 96 KB
 - Instructions: 64 KB
- **Flash:**
 - 512KB to 4KB up to 16KB (40 Mhz can be doubled to 80Mhz)
- **BUS:**
 - SPI (communicate with the falsh memory)
 - I2C
 - I2S: interface with DMA
- **I/O**
 - 16 pins GPIO
 - 1 bits ADC(Analog to Digital Converter 10 bits precision (0 – 1023 values)) 0 up to 1V.
- **WIFI IEEE 802.11 b/g/n**
 - Authentication: WEP, WPA/WEP2
 - Opened networks
- **Built-in temperature sensor**
- **Electrical features**
 - Input voltage: **3.3v**
 - Input current
 - total suhtdow: **0.5 uA**
 - Transmit 802.11b, CCK 1Mbps, POUT=+19.5dBm: **215 mA**
 - Max output current: **12 mA**

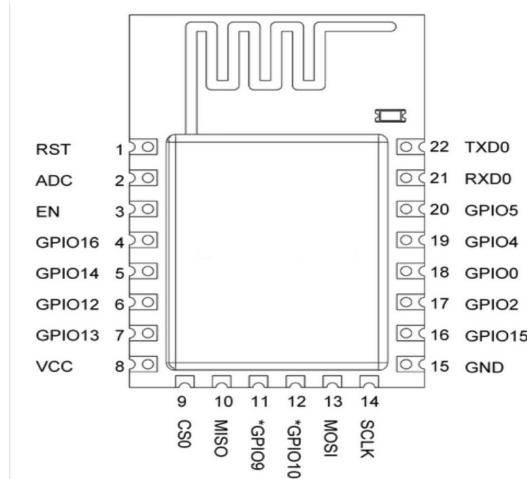
1.1.2. SDK

ESP8266 Arduino core permet à la carte ESP8266 WIFI et au CPU d'être programmés comme un composant Arduino avec des bibliothèques adaptées aux protocoles suivants :

- TCP
- UDP
- HTTP

- DNS
- SD card
- I2C
- SPI
- ...

1.1.3. Pins description



- **RST** - Resets the module (Reboot)
- **ADC** - Analog to Digital Conversion. Used for reading analog values.
- **EN** - Chip enable
- **GPIO16** - GPIO16; Can be used to wake up the chipset from deep sleep
- **GPIO14** - General Purpose Input and Output 14; HSPI_CLK
- **GPIO12** - General Purpose Input and Output 12; HSPI_MISO
- **GPIO13** - General Purpose Input and Output 13; HSPI_MOSI; UART0_CTS
- **VCC** - 3.3V power supply (VDD); **IMPORTANT! The device's operating voltage is between 3.0V and 3.6V. 5Volts may kill the device!**
- **CS0** - Chip selection
- **MISO** - Slave output Main input
- **GPIO9**
- **GPIO10**
- **MOSI** - Main output slave input
- **SCLK** - Clock
- **GND** - Ground
- **GPIO15**
- **GPIO2**
- **GPIO0** - This port is used for programming. Connecting it to GND puts the device in FLASH mode, where you can flash new firmware and upload a programm (also called Sketch).
- **GPIO4**
- **GPIO5**
- **RXD** - UART0_RXD - UART TTL Receive Data pin. Used for programming, flashing and for communication with the module. It is also GPIO3.
- **TXD** - UART0_TXD - UART TTL Transmit Data pin. It is also GPIO1.

1.2. Capteur de courant SCT013

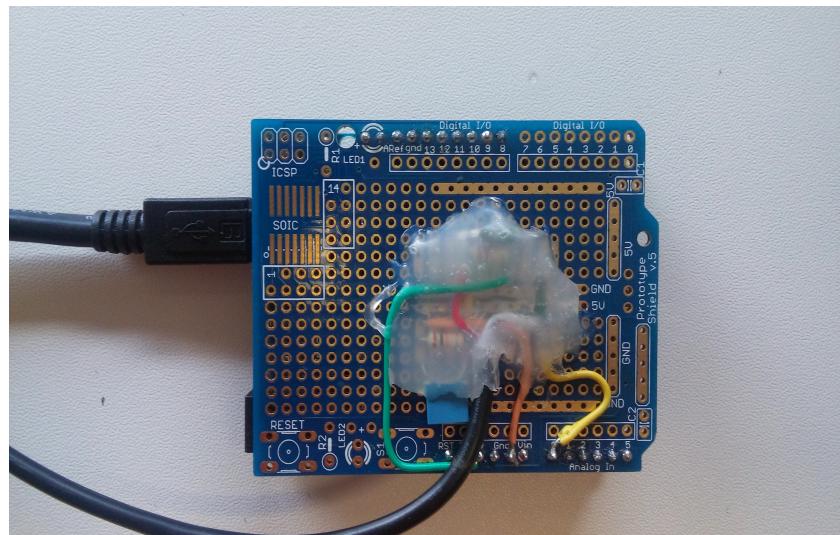


Caractéristiques techniques:

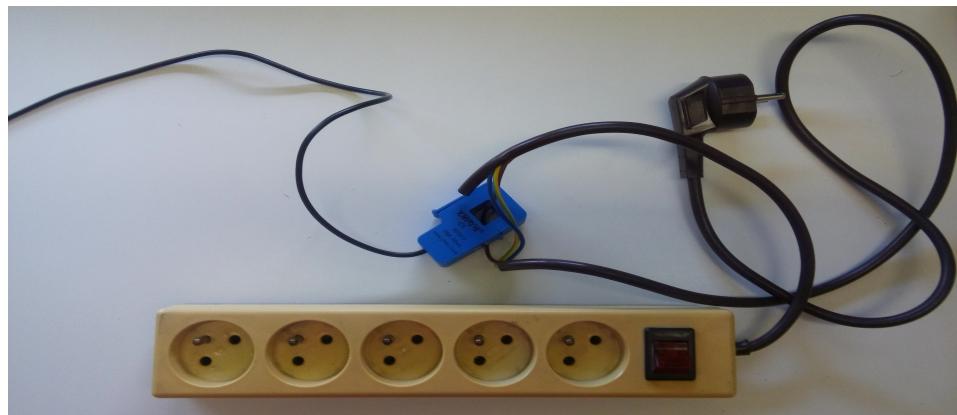
1. Mesure en entrée un courant Max de 100 A
2. Fournie en sortie un courant max de 50 mA après une transformation de facteur 1:2000
3. Le courant est transformé en tension max de 5 V à l'aide d'un circuit RC monté sur la carte prototype shield v.5, ainsi l'ESP8266 peut la mesurer

Prototype shield v.5

Utilisé pour interconnecter le capteur à l'ESP8266

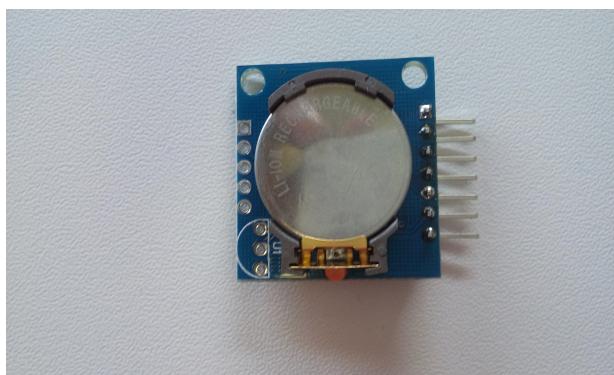
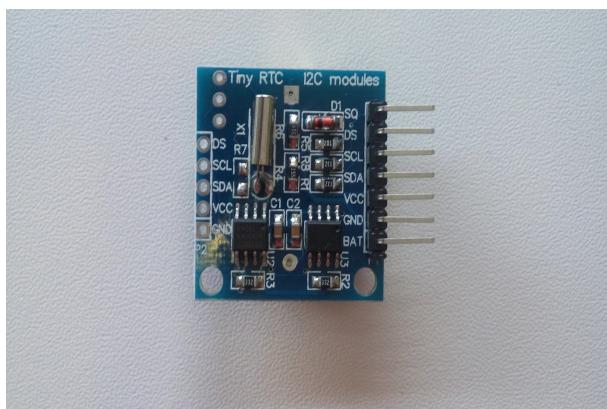


Les appareils dont on veut mesurer la puissance consommer doivent être branchés sur la multiprise qui elle-même est connectée au secteur.

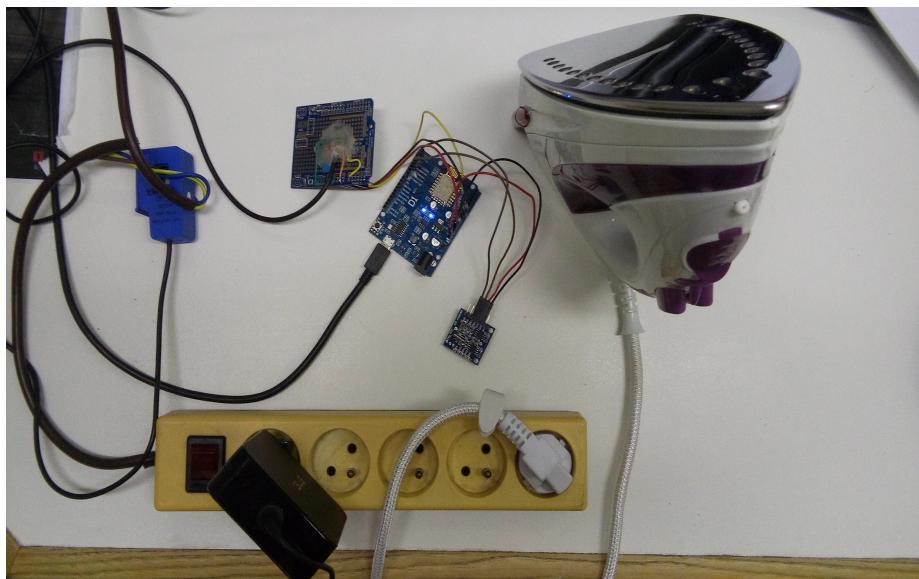


1.3. Horloge Tiny RTC

Elle permet d'avoir la date et l'heure en temps réel utilisée dans ce cas pour horodater les mesures.



1.4. Réalisation du montage



Nous utilisons des fils femelle mal.

1. Connexion de l'horloge RTC à la carte ESP8266
 - pin SDA de RTC ----> pin D14/SDA/D4 de ESP8266
 - pin SCL de RTC ----> pin D15/SCL/D3 de ESP8266
 - pin GND de RTC ----> pin GND de ESP8266
 - pin VCC de RTC ----> pin 3,3 V de ESP8266
2. Connexion du capteur SCT013 à l'ESP8266 à travers la carte prototype shield v.5
 - pin 5 V de prototype shield v.5 ----> pin 5 V de ESP8266
 - pin GND de prototype shield v.5 ----> pin GND de ESP8266
 - pin A0 de prototype shield v.5 ----> pin A0 de ESP8266

1.5. Téléversement du code dans l'ESP8266

1. Connecter la carte à l'ordinateur à travers un cable USB qui ne sert pas que de chargeur mais peut transmettre des données
2. Sur ubuntu taper la commande: **sudo chown nom_utilisateur /dev/ttyUSB0**

3. Dans l'IDE Arduino, cliquer sur upload à l'extreme gauche de la fenêtre de l'IDE

NB: avant de téléverser le code veuillez insérer vos paramètres de connexion internet **SSID** et **mot de passe** de Wifi sinon pour que les mesures puissent être envoyées.

Mesures envoyées par le capteur au serveur dans Docker

{

numValue: la puissance consommée,
time: la date et l'heure courante en secondes depuis le 01/01/1970 à 00:00:00,
device_id: l'ID de l'ESP8266
region: la position géographique du capteur

}

2. Configuration serveur

2.1. Principe

Les datagrammes udp des mesures sont capturés sur un nœud d'entrée node-red du serveur backend avant d'être ensuite transformés dans un pipeline de traitements node-red ayant pour responsabilité de vérifier les informations collectées et les mettre en forme préalablement à leur intégration dans InfluxDB.

Le stockage des mesures de courant se fait dans une base temporelle InfluxDB.

L'outil de visualisation des profils de consommation est Grafana qui permet de mettre en forme de tableau de bord les données stockées dans InfluxDB.

2.2. Configuration générale de Docker

Suite à différents problèmes techniques rencontrés sur la machine Windows de développement du backend pour réceptionner des datagrammes sur un port udp dans des conteneurs Docker, nous avons pris la décision d'automatiser au maximum le déploiement de notre configuration Docker en prévision de son déploiement sur une machine Ubuntu.

Cela nous a conduit :

- A construire une image Docker spécifique pour installer notre conteneur node-red afin d'automatiser l'installation du plugin node-red/InfluxDB ainsi que l'import de notre flux node-red et l'exposition du port udp 8123,
- A utiliser docker-compose pour décrire via un fichier les trois services Docker dont nous avons besoins ainsi que leur interdépendance.

Le fichier Dockerfile de construction de notre image node-red contient la configuration suivante :

```
FROM nodered/node-red-docker:latest
RUN npm install node-red-contrib-InfluxDB
COPY flows.json /data/
CMD ["npm", "start", "--", "--userDir", "/data"]
EXPOSE 8123/udp
```

Le fichier flows.json contient un export de notre flux node-red (effectué après mis au point de celui-ci depuis le client web node-red)

Nous utilisons pour docker-compose.yml cela docker-compose

```
version: '2'
services:
  red:
    build: ./red
    ports:
      - "1880:1880"
      - "8123:8123/udp"
    links:
      - bd
  bd:
    image: "tutum/InfluxDB"
    ports:
      - "8083:8083"
      - "8086:8086"
    environment:
      - ADMIN_USER=franck
      - INFLUXDB_INIT_PWD=fred
      - PRE_CREATE_DB=power
    volumes:
      - ./bd:/var/lib/InfluxDB
  graph:
    image: "grafana/grafana"
    ports:
      - "3000:3000"
    links:
      - bd
```

2.3. Configuration de la base de données InfluxDB

Administration web depuis l'adresse 192.168.99.100 :8083

Une base de données ‘power’ a été créée dès le montage Docker avec la variable d’environnement PRE_CREATE_DB positionnée dans le docker-compose.yml. Nous avons également paramétré dans ce fichier un changement de l’utilisateur administrateur et de son mot de passe pour éviter le profil admin/admin par défaut.

Une mesure powerval collectera toutes les mesures de courant au cours du temps. Les tags ‘device’ et ‘region’ ont été rajoutés pour identifiés l’id unique pince et la région dans laquelle elle est utilisée.

Pour faciliter le déploiement Docker automatique et dans l'attente d'un passage à l'échelle, nous avons retiré deux fonctionnalités installées initialement en console depuis le conteneur :

- 1> La modification de la politique de rétention selon nos besoins et les capacités de stockage avec l'instruction (ici 1 an) :

```
CREATE RETENTION POLICY "un_an" ON power DURATION 365d REPLICATION 1 DEFAULT;
```

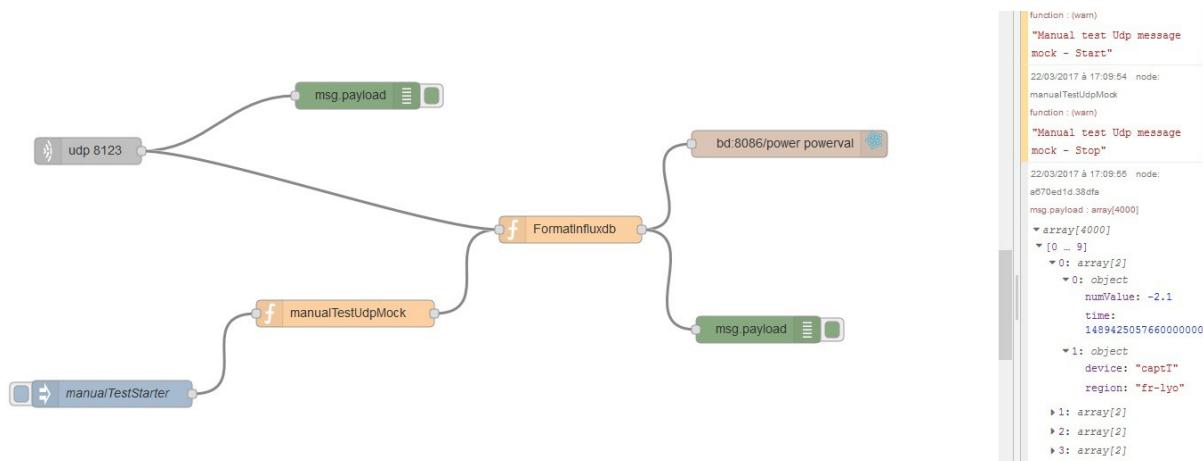
- 2> L'ajout d'un utilisateur avec des privilèges moindres que l'administrateur pour l'intégration en flux depuis node-red:

```
CREATE USER yyy WITH PASSWORD 'xxx';
GRANT ALL ON power TO yyy;
```

2.4. Configuration de node-red

Administration web depuis l'adresse 192.168.99.100 :1880

En point d'entrée, les payloads attendus des datagrammes udp sur le port 8123 contiennent des tableaux d'objets ou un objet de la forme {numValue :1.223, time : 1489425057660000000, device :'captxxx', device :'fr-gre'}



La fonction FormatInfluxDB formate les données au format attendu par InfluxDB (en isolant notamment dans 2 tableaux séparés les tags des autres données numériques de mesure et de temps).

```

if (!msg || !msg.payload) {
  node.warn("Empty payload");
  return null;
}

//One item formating function
const formatOnItem = function( info ) {
  if (!info.numValue || !info.time) {
    node.warn("Incomplete item "+JSON.stringify(info));
    return null;
  }
  let deviceld = !info.device ? 'captxx' : info.device;
  let region = !info.region ? 'fr-xx' : info.region;
}

```

```

let item = [];
item.push({
    numValue:info.numValue //,
    // time:info.time //
});
item.push({
    device:deviceId,
    region:region
});
//node.warn("Item : "+JSON.stringify(item));
return item;
};

const sentInfo = msg.payload;
if (Array.isArray(sentInfo)) {
    let arrayReturned = [];
    for (let i = 0; i < sentInfo.length; i++){
        const oneItem = formatOneItem(sentInfo[i]);
        if (oneItem !== null) {
            arrayReturned.push(oneItem);
        }
    }
    return {payload : arrayReturned};
} else {
    let stringItem=msg.payload.substring(0,msg.payload.indexOf('}')+1);
    sentInfo = JSON.parse(stringItem);
    let arrayReturned = [];
    const oneItem = formatOneItem(sentInfo);
    if (oneItem !== null) {
        arrayReturned.push(oneItem);
    }
    return {payload : arrayReturned};
}
return null;

node.warn("Manual test Udp message mock - Start");
let content = [];
for (let i = 0; i < 3; i++){
    content.push({
        numValue:-2.1+(i%100)/25,
        time:148942505766000000+1000*i,
        device:'captT',
        region:'fr-lyo'
    });
}
node.warn("Manual test Udp message mock - Stop");
return { payload: content };

graph:
image: "grafana/grafana"
ports:
- "3000:3000"
links:
- bd

```

Le bouton manualTestStarter et la fonction manualTestUdpMock ont pour fonction de simuler la génération de 12000 mesures dans la base au format attendu par la datagrammes.

```

node.warn("Manual test Udp message mock - Start");
let content = [];
for (let i = 0; i < 12000; i++){
    content.push({

```

```

        numValue:-2.1+(i%100)/25,
        time:1489425057660000000+1000*i,
        device:'captT',
        region:'fr-lyo'
    });
}
node.warn("Manual test Udp message mock - Stop");
return { payload: content };

```

Difficultés rencontrées : nous n'avons pas pu intégrer correctement l'horodatage transmis par l'horloge Tiny RTC et avons dû désactiver l'information transmise (en jaune ci-dessus) et laissé Influxdb horodater par lui-même les données.

2.5. Configuration de Grafana

Administration web depuis l'adresse 192.168.99.100 :3000 (profil : admin/admin)

La base 'power' est rajoutée manuellement dans la configuration 'Data Sources' du menu principal (Name : bd ; type : InfluxDB ; Url : <http://bd:8086> ; InfluxDB Details – Databse : power ; Useryyy ; Passwordxxx).

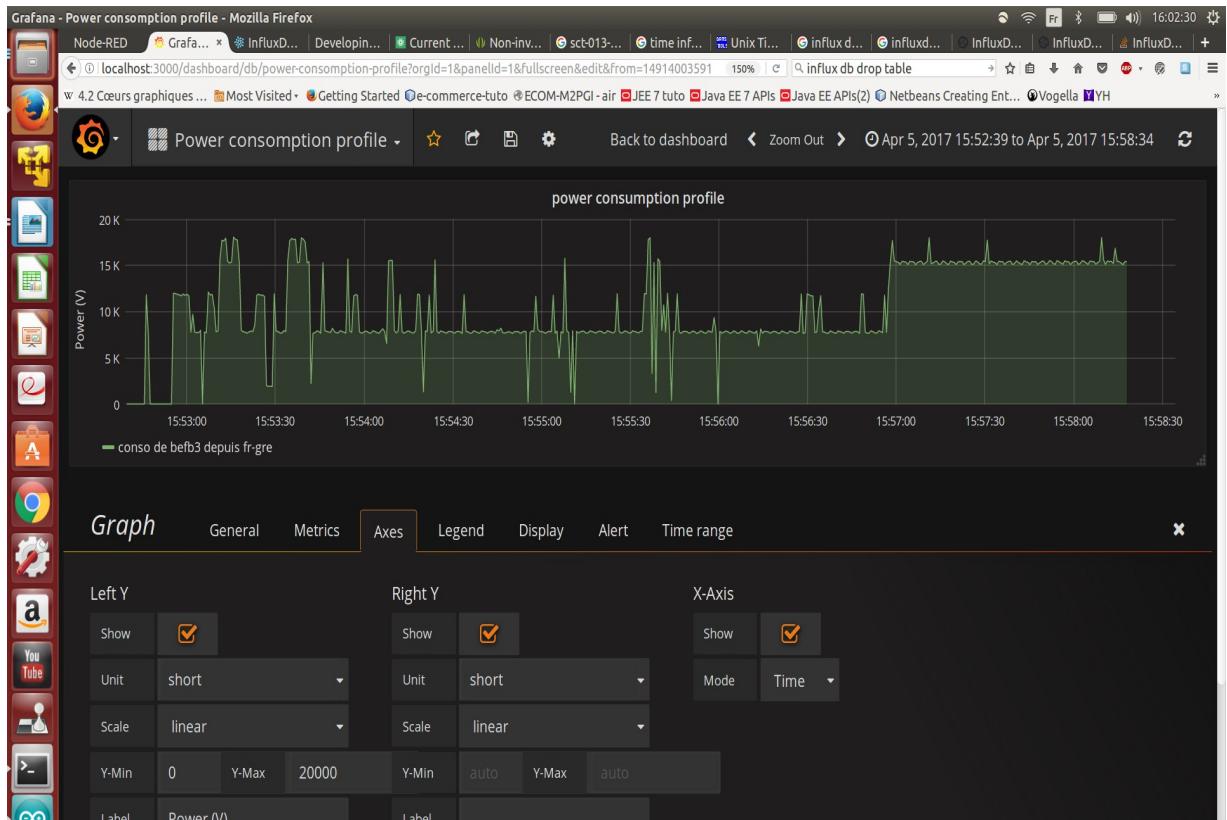
Le tableau de bord a été développé puis exporté dans un fichier dashPowerProfile.json en format json. Dans le nouveau conteneur déployé, il est importé depuis le menu de gestion des tableaux de bord de Grafana.

Le résultat suivant a été obtenu après génération de la fonction de test node-red :

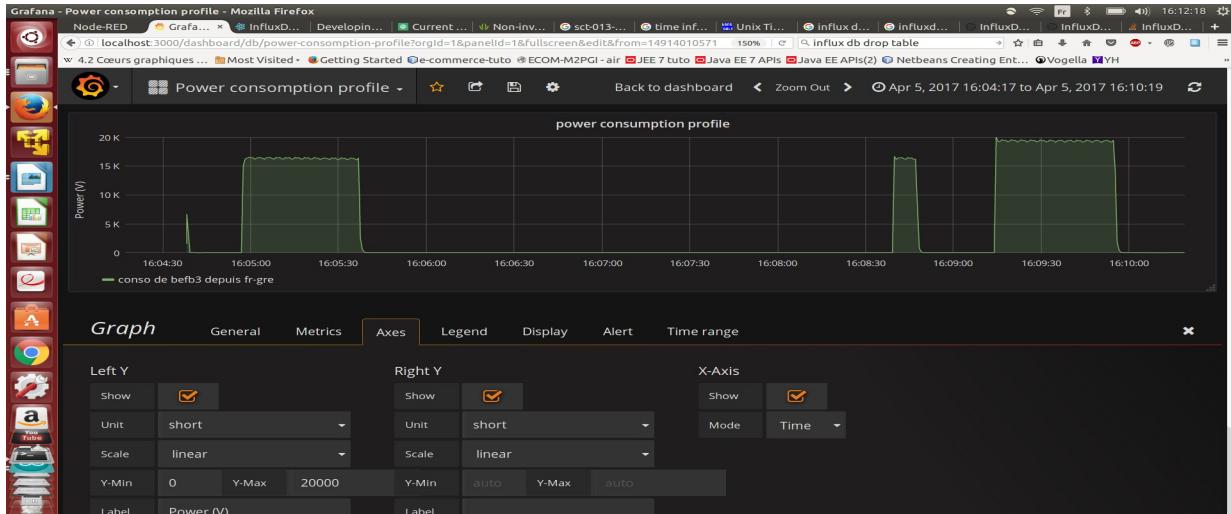


3. Profils de consommation observés

3.1. Profil d'une consommation d'un sèche-cheveux



3.2. Profil de consommation d'un fer à repasser



Voici une capture des données enregistrées dans influxdb

time	device	numValue	region	value
2017-04-05T14:04:38.220971316Z	"befb3"	1575.37	"fr-gre"	
2017-04-05T14:04:38.221510271Z	"befb3"	6669.36	"fr-gre"	
2017-04-05T14:04:39.283354317Z	"befb3"	372.84	"fr-gre"	
2017-04-05T14:04:39.308223888Z	"befb3"	91.43	"fr-gre"	
2017-04-05T14:04:39.796188522Z	"befb3"	36.82	"fr-gre"	
2017-04-05T14:04:40.452145129Z	"befb3"	33.51	"fr-gre"	
2017-04-05T14:04:41.135374935Z	"befb3"	31.3	"fr-gre"	
2017-04-05T14:04:41.762838471Z	"befb3"	30.57	"fr-gre"	
2017-04-05T14:04:42.418567976Z	"befb3"	30.07	"fr-gre"	
2017-04-05T14:04:43.075299976Z	"befb3"	28.97	"fr-gre"	
2017-04-05T14:04:43.732035785Z	"befb3"	31.01	"fr-gre"	
2017-04-05T14:04:44.385263395Z	"befb3"	37.59	"fr-gre"	
2017-04-05T14:04:45.041126678Z	"befb3"	33.69	"fr-gre"	