

# Effevee's Weerstation

(foto van meetstation + screenshot van InfluxDB dashboard)

Alle documentatie en code van dit project kan je hier downloaden. De code is ruim voorzien van commentaar en uitleg. Mocht je nog met vragen zitten, stuur dan gerust een email naar effevee AT gmail DOT com.

Enjoy,

Effevee.

# Inhoudsopgave

Beschrijving.....	4
Materiaal.....	4
Meetstation.....	4
Backend.....	4
Meetstation.....	5
Microcontroller.....	5
Sensoren.....	6
AM2320 temperatuur- en vochtigheidssensor.....	6
BMP180 luchtdruk- en temperatuursensor.....	6
BH1750 licht intensiteitssensor.....	6
Voeding.....	7
Lithium-Ion 18650 batterij 2500 mAh.....	7
Zonnepaneel 6V 100mA 0,6W – 90x60 mm.....	7
TP4056 lithium batterij laad- en beschermingsmodule.....	8
HT7333 3,3V spanningsregelaar.....	8
Elektrisch schema.....	9
Microcontroller.....	9
Voeding.....	9
Sensoren.....	10
Batterijspanning monitoring.....	10
PCB.....	11
Behuizing.....	12
Backend.....	13
Raspberry Pi.....	13
Case.....	13
USB 3.1 Flash Drive.....	13
Voeding.....	14
InfluxDB database.....	15
Data structuur.....	15
Velden.....	15
Lijn protocol.....	15
Software.....	16
Meetstation.....	16
MicroPython firmware.....	16
Download firmware.....	16
Wissen ESP32 flash.....	16
Uploaden MicroPython naar ESP32 flash.....	16
Testen MicroPython op ESP32.....	16
MicroPython code.....	17
Modules.....	17
Configuratie.....	17
Functies.....	18
Backend.....	19
Raspberry Pi OS.....	20
Docker.....	22
Docker-Compose.....	23
Docker network.....	23
Docker containers.....	24

Eclipse Mosquitto.....	24
Influxdb.....	25
Telegraf.....	28
InfluxDB dashboard.....	30
Temperatuur.....	31
Vochtigheid.....	32
Luchtdruk.....	33
Lichtsterkte.....	34
Batterijspanning.....	35
Temperatuur nu.....	36
InfluxDB monitoring.....	37
Controle batterijspanning.....	37
Controle meetstation.....	39
SendGrid e-mail service.....	40
Batterij waarschuwing e-mail.....	42
Meetstation waarschuwing e-mail.....	43
Bijlagen.....	45
Datasheets.....	45
Referenties.....	45

# Beschrijving

Effevee's weerstation bestaat uit een buiten meetstation gebaseerd op een [ESP32](#) microcontroller met sensoren voor temperatuur, vochtigheid, luchtdruk en licht intensiteit. Het autonome meetstation wordt gevoed door een lithium 18650 batterij met aangepaste 3,3V spanningsregelaar. De batterij wordt met een 6V 100mA 0,6W zonnepaneeltje en een TP4056 controller veilig opgeladen. De sturing van het meetstation is geprogrammeerd in [MicroPython](#).

De meetresultaten inclusief de batterijspanning worden via WiFi doorgegeven naar de [Raspberry Pi](#) backend [mosquitto](#) MQTT broker. Deze worden vervolgens met [telegraf](#) doorgestuurd naar een [influxdb](#) database. De meetgegevens worden tenslotte gevisualiseerd in een influxdb dashboard (Chronograf) in de browser. Als de batterijspanning te laag wordt stuurt influxdb (Kapacitor) een email. Alle nodige software op de backend wordt met [docker](#) containers geïnstalleerd.

## Materiaal

### Meetstation

- DOIT ESP-WROOM-32 ontwikkelbordje
- AM2320 temperatuur- en vochtigheidssensor
- BMP180 luchtdruk- en temperatuursensor
- BH1750 licht intensiteitssensor
- Lithium-Ion 18650 batterij 2500 mAh + batterijhouder
- Zonnepaneel 6V 100mA 0,6W – 90x60 mm
- TP4056 lithium batterij laad- en beschermingsmodule + 10k weerstand
- HT7333 3,3V spanningsregelaar + 2 100µF/16V elektrolytische condensatoren
- Batterijspanning monitoring : weerstanden 27k + 100k, condensator 100nF

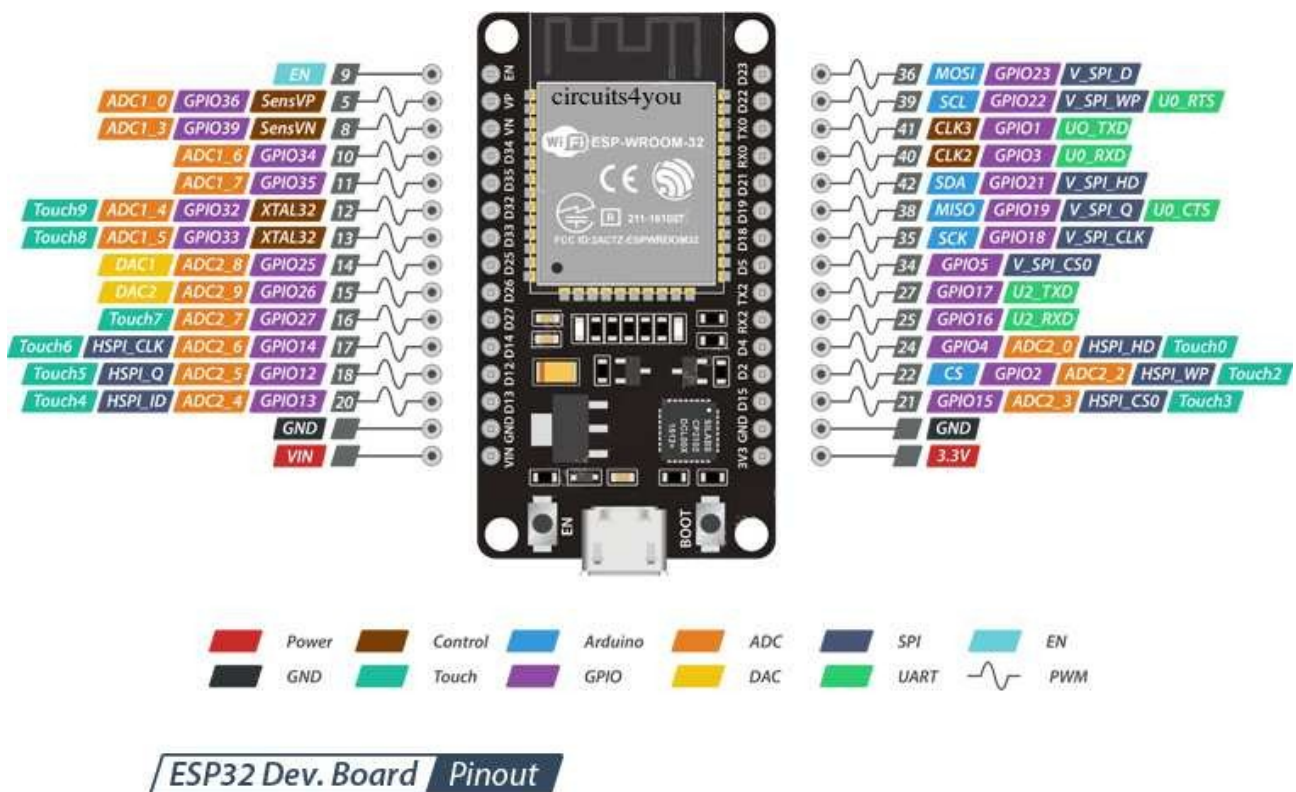
### Backend

- Raspberry Pi 4 Model B 2GB
- Aluminium passieve koeler behuizing
- SanDisk Ultra Fit USB 3.1 Flash Drive
- Originele Pi USB-C 3A voeding

# Meetstation

## Microcontroller

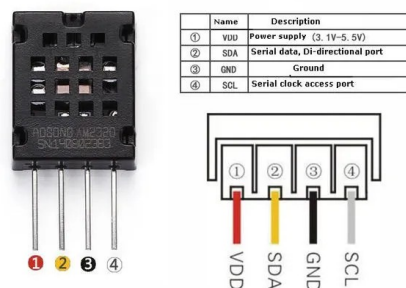
We maken gebruik van het DOIT ESP-WROOM-32 ontwikkelbordje dat vlot verkrijgbaar is. De ESP32 heeft 2 processors aan boord, communiceert via WiFi en BLE (bluetooth low energy). De klok gaat tot 240 MHz. Er is 4 MB RAM geheugen en allerlei protocollen worden ondersteund zoals capacitive touch, ADC, DAC, UART, SPI, I2C, ...



## Sensoren

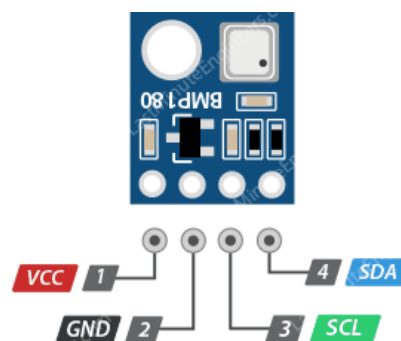
### AM2320 temperatuur- en vochtigheidssensor

De AOSONG AM2320 compacte sensor is de opvolger van de verouderde DT11/DT22 sensoren. De sensor is nauwkeurig, vergt weinig stroom en is simpel aan te sturen via I2C op adres 0x5C.



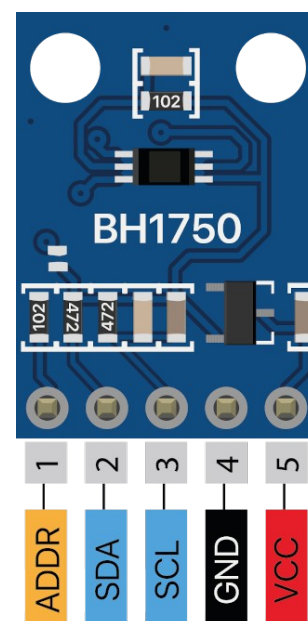
### BMP180 luchtdruk- en temperatuursensor

De Bosch BMP180 sensor levert luchtdruk meetwaarden ten opzichte van zeeniveau en kan gekalibreerd worden door parameters in de EEPROM van de module. Deze sensor is eveneens zuinig en aan te sturen via I2C op adres 0xD0.



### BH1750 licht intensiteitssensor

De Mouser BH1750 sensor wordt eveneens aangestuurd via I2C op adres 0x23, heeft een hoge resolutie (1-65235 lux) en is energie zuinig met zijn powerdown functie.

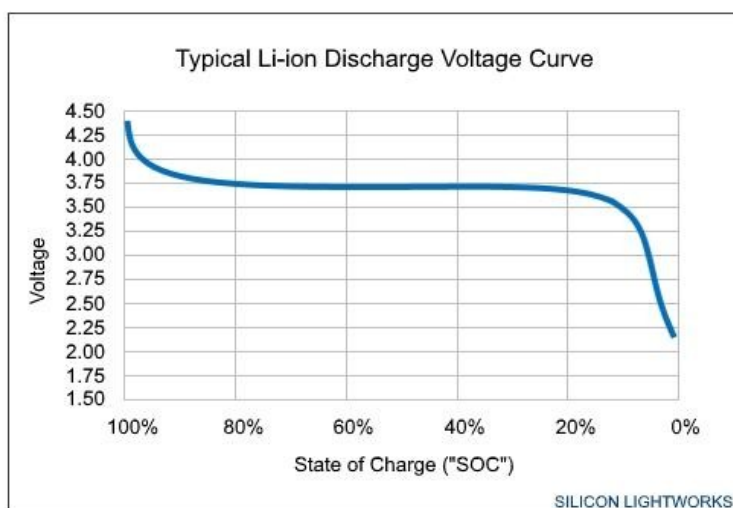


## Voeding

De voeding bestaat uit volgende onderdelen :

### Lithium-Ion 18650 batterij 2500 mAh

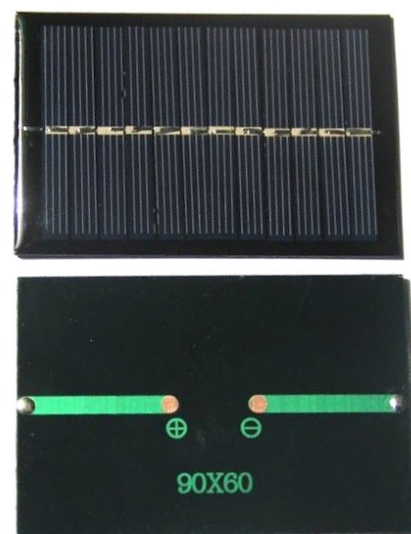
Een lithium ion batterij heeft een hoge energiedichtheid en een geringe zelfontlading. Bovendien heeft ze geen last van geheugeneffect (capaciteitsvermindering bij opladen voordat ze volledig ontladen is). De nominale spanning (bij 50% lading) is 3,7V. Bij volledig opladen moet de spanning beperkt worden tot 4,2V. Het is af te raden om de cel onder de 2,4V te laten ontladen. De cel kan ongeveer 2000 keer volledig geladen/ontladen worden. Niet gebruikte cellen worden best op hun nominale spanning bij kamertemperatuur gestockeerd.



Hiernaast zie je een typische ontlad curve van een lithium ion cel. De maximale spanning van 4,2V verlaagt heel snel naar de nominale spanning van 3,7V maar die blijft dan heel lang stabiel tot ongeveer 10% van de lading. Daarna gaat het heel snel bergaf tot de minimum spanning.

### Zonnepaneel 6V 100mA 0,6W – 90x60 mm

Dit paneeltje bevat 12 cellen die elk ongeveer 0,5V kunnen leveren.



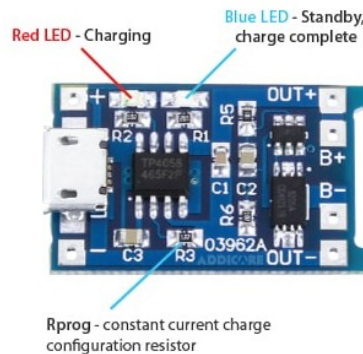
## TP4056 lithium batterij laad- en beschermingsmodule

Deze [TP4056 beschermingsmodule](#) bevat een TP4056 chip die een 3,7V lithium batterij oplaadt via een constante stroom/constante spanning methode (CC/CV) tot 4,2V.

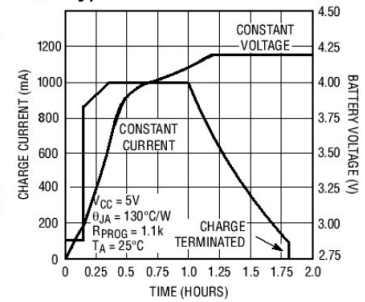
De DW01A chip houdt de spanning van de batterij in de gaten en stuurt de FS8204A transistor om de load af te schakelen indien de spanning onder de 2,4V dreigt te gaan. Die blijft afgeschakeld tot de spanning terug boven de 3,0V is. Daarnaast zorgt deze beschermingschip er ook voor dat de batterij tot maximaal 4,2V geladen wordt en dat de ontlaad stroom onder de 3A blijft.

Als de spanning onder de 2,9V ligt wordt de batterij met een verlaagde stroom (130mA) opgeladen tot 2,9V. Daarna wordt de stroom lineair opgevoerd tot de maximum laadstroom die bepaald wordt door de waarde van de weerstand R3 op de module (standaard 1,2k = 1A)

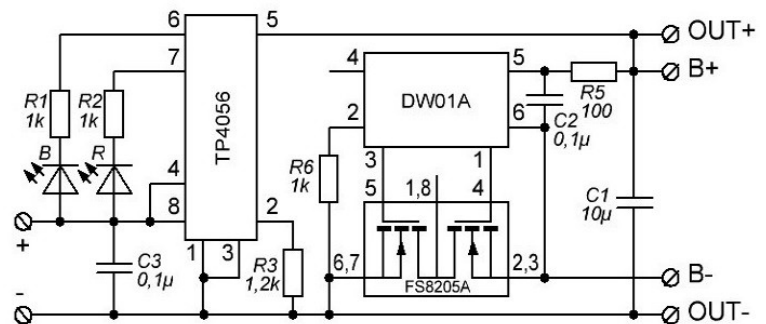
Het zonnepaneeltje levert maximaal 100 mA stroom, dus kunnen we de TP4056 module best aanpassen voor deze stroom. Hiervoor heb ik een 10k gesoldeerd op R3 zodat de laadstroom beperkt wordt tot 130mA. Het zonnepaneel levert maximaal 6V en kan dus rechtstreeks op de input van de TP4056 module worden aangesloten (max 8V input)



Complete Charge Cycle (1000mAh Battery)



TP4056 1A Battery Charger Module with Protection Circuit

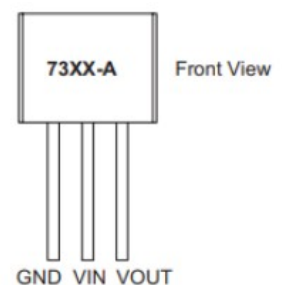


Rprog Current Setting

R <sub>PROG</sub> (k)	I <sub>BAT</sub> (mA)
10	130
5	250
4	300
3	400
2	580
1.66	690
1.5	780
1.33	900
1.2	1000

## HT7333 3,3V spanningsregelaar

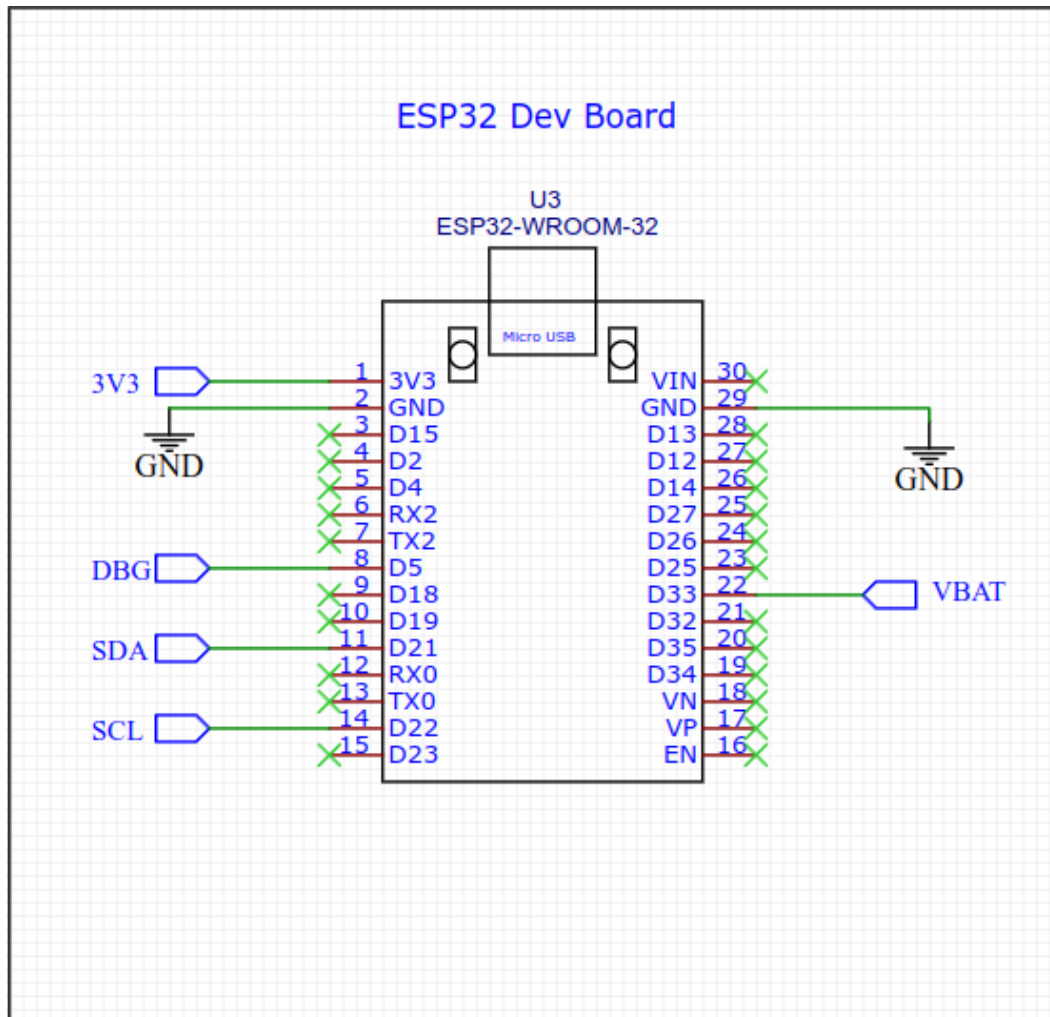
De gewone spanningsregelaars hebben een dropout spanning van ongeveer 1V. De regelaar werkt enkel als  $V_{in} > V_{out} + V_{dropout}$ . In ons geval zou dat betekenen dat de ingangsspanning hoger dan 4,3V moet zijn om onze gewenste uitgangsspanning van 3,3V te bekomen. Onze lithium cel heeft slechts een maximale spanning van 4,2V en een nominale spanning van 3,7V. Dus is een gewone spanningsregelaar onbruikbaar. We hebben een LDO (low dropout) spanningsregelaar zoals de HT7333 nodig die maar een dropout spanning van 90mV heeft. De regelaar blijft dus werken tot een ingangsspanning van ongeveer 3,4V.



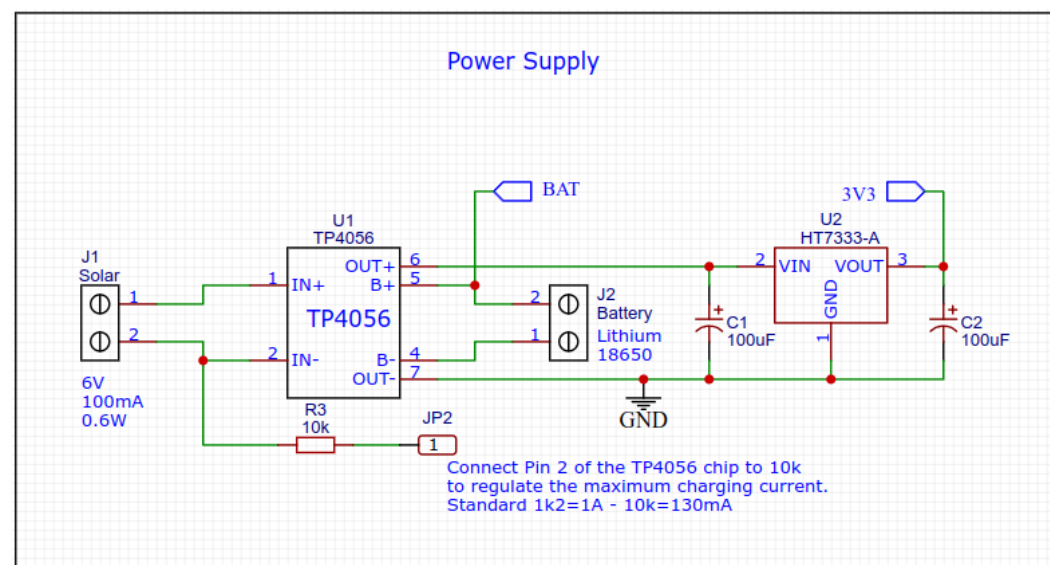


## Elektrisch schema

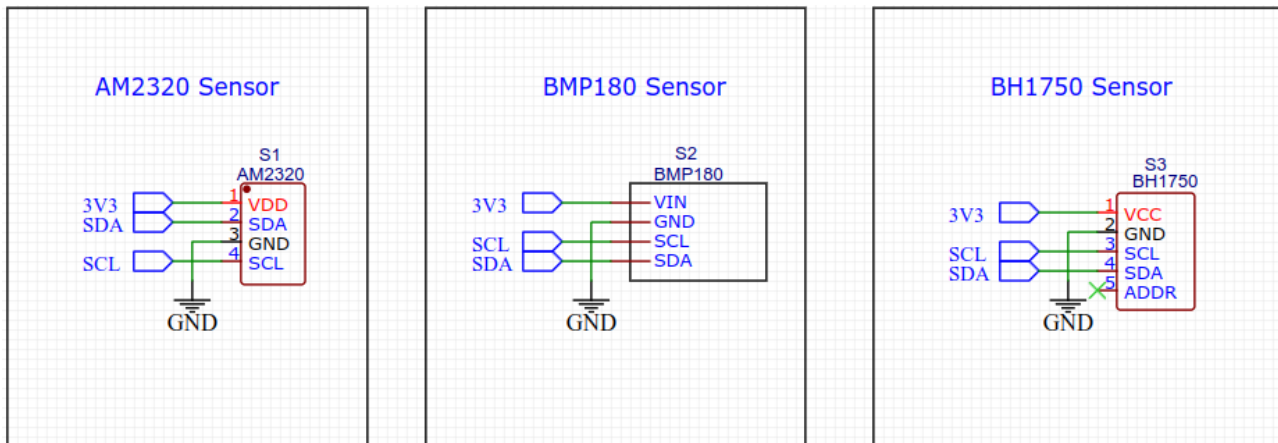
### Microcontroller



### Voeding



## Sensoren



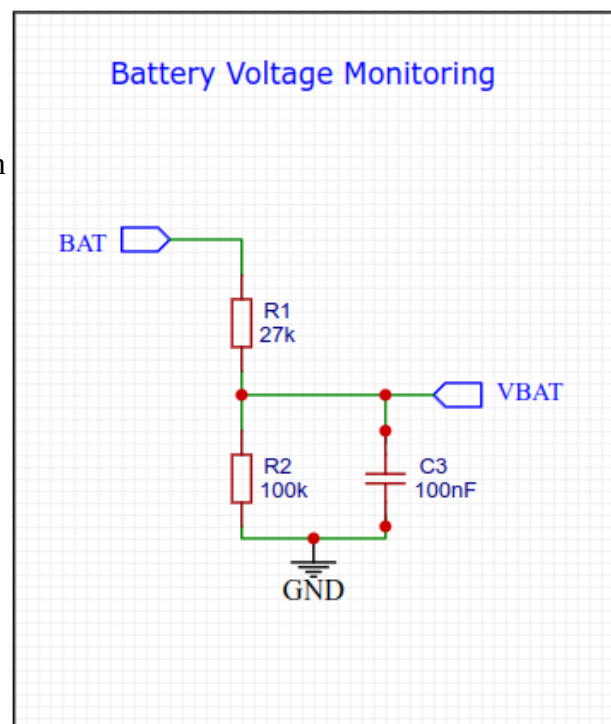
## Batterijspanning monitoring

We lezen de spanning van de Li-Ion 18650 batterij uit via de GPIO 33 pin van de ESP32 microcontroller. Daar de spanning op deze pin maximaal 3,3V mag zijn, moeten we gebruik maken van een spanningsdeler R1 en R2.

$$V_{\text{BAT}} = \text{BAT} * R2 / (R1 + R2)$$

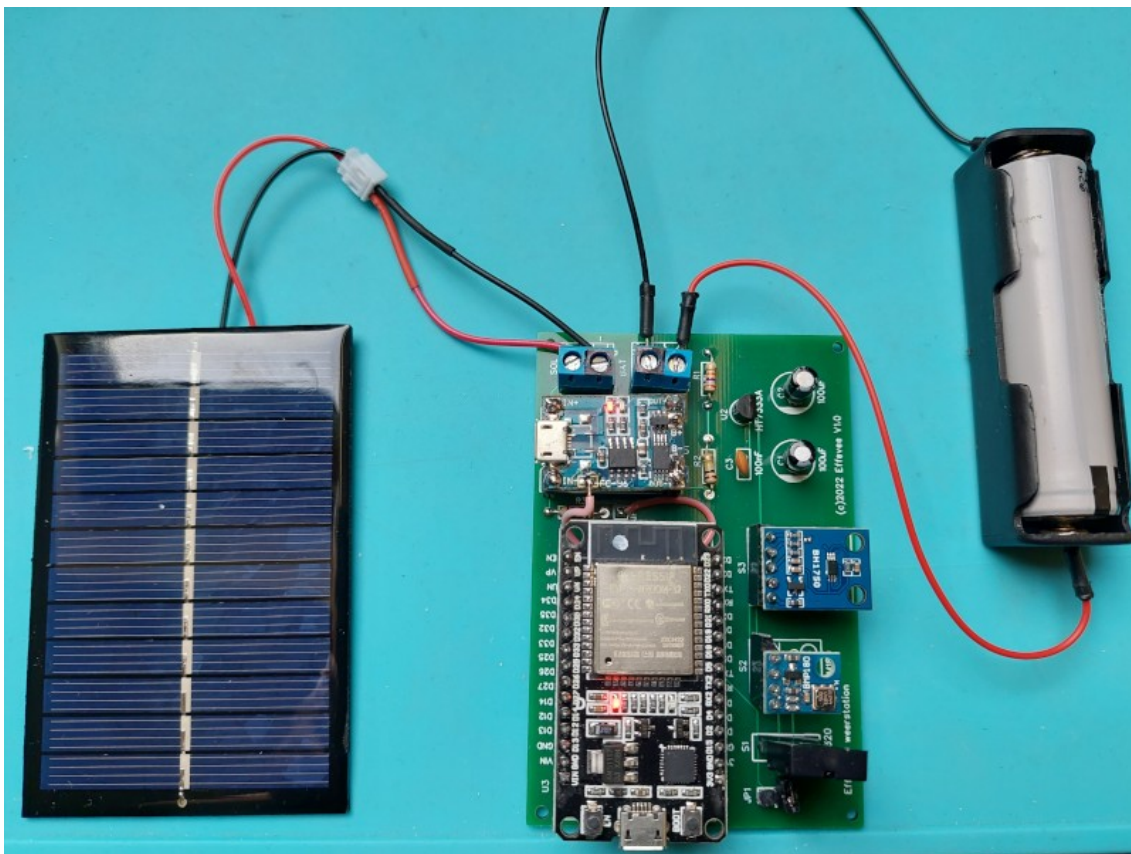
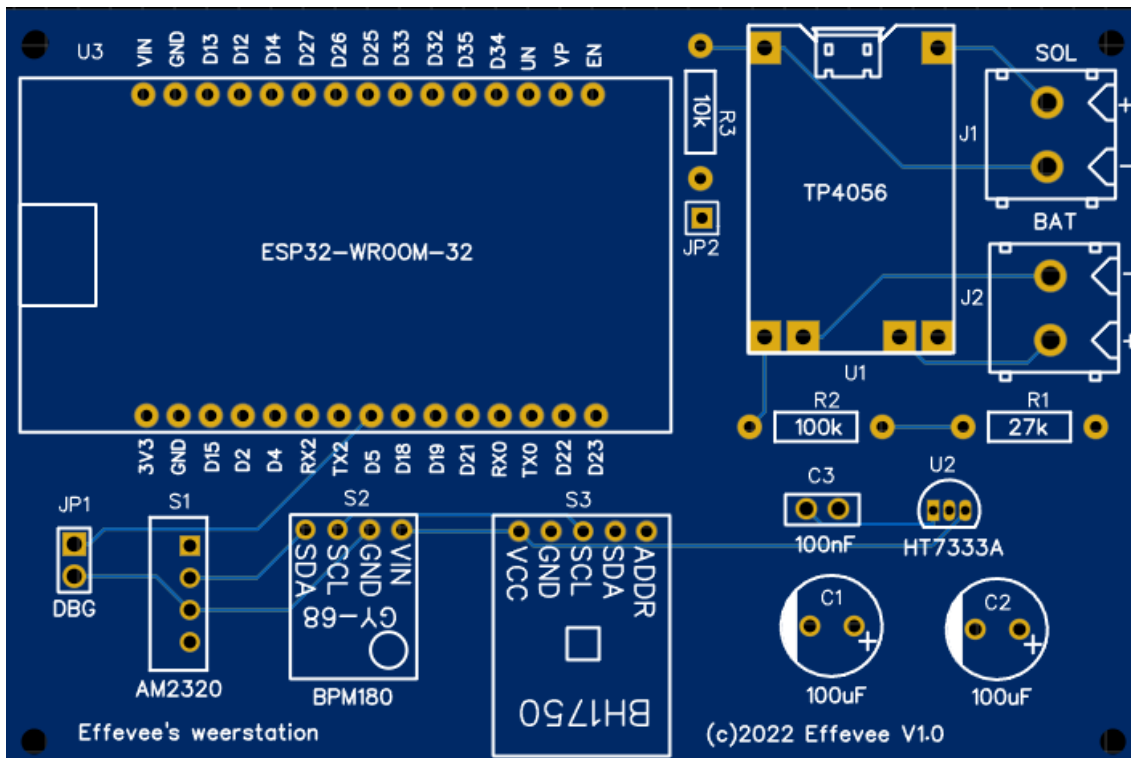
$$\begin{aligned} V_{\text{BAT}} &= 4,2\text{V} * 100\text{k} / (100\text{k} + 27\text{k}) \\ &= 4,2\text{V} * 100\text{k} / 127\text{k} \\ &= 4,2\text{V} * 0,79 \\ &= 3,3\text{V} \end{aligned}$$

De condensator C3 vangt eventuele spanningspieken op om de ESP32 pin te beschermen.



## PCB

Ik heb een printplaatje ontworpen met de [EasyEDA Online Editor](#) en laten produceren door [JLCPCB](#). De gerber bestanden kan je ook vinden op de github.



## Behuizing

Ik heb het geheel ingebouwd in [deze behuizing](#) die ik heb geprint met mijn 3D printer.

# Backend

## Raspberry Pi

Ik gebruik hiervoor een [Pi 4 Model B 2GB](#)



## Case

Die is uitgerust met een [aluminium passieve koeler](#) behuizing



## USB 3.1 Flash Drive

Het OS wordt op een [SanDisk Ultra Fit USB 3.1 flash drive](#) van 16GB gebrand. Deze is heel wat robuuster dan een microSD kaartje, vooral ook omdat de software heel wat lees- en schrijf acties zal genereren.



## Voeding

Ik gebruik de [originele Pi USB-C 3A](#) voeding



# InfluxDB database

De gegevens van het meetstation en de OpenWeatherMap voorspelling zullen we opslaan in een Influxdb2 database.

## Data structuur

bucket (database)	weatherdata	
measurements (tabellen)	<i>forecasts</i>	<i>actuals</i>
tags (index velden)	<i>source='OpenWeatherMap'</i> <i>location='Ingelmunster,BE'</i>	<i>source='weerstation'</i> <i>location='Ingelmunster,BE'</i>
fields (velden)	<i>ow_temp</i> (float) <i>ow_hum</i> (float) <i>ow_pres</i> (float)	<i>ac_temp</i> (float) <i>ac_hum</i> (float) <i>ac_pres</i> (float) <i>ac_lum</i> (float) <i>ac_vbat</i> (float)
timestamps (tijd veld)	<i>time</i> (timestamp)	<i>time</i> (timestamp)

## Velden

- \*\_temp : temperatuur in °C
- \*\_hum : vochtigheid in %
- \*\_pres : luchtdruk in hPa
- ac\_lum : lichtintensiteit in lux
- ac\_vbat : batterij spanning in Volt

## Lijn protocol

Voor het schrijven van data naar de database moet een speciaal formaat worden gebruikt.

```
<measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]]  
<field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>]
```

Vertaald naar ons geval geeft dit de 2 volgende lijnen :

```
forecasts,source="...",location="..." ow_temp=...,ow_hum=...,ow_pres=...  
actuals,source="...",location="..." ac_temp=...,ac_hum=...,ac_pres=...,ac_lum=...,ac_vbat=...
```

We geven geen timestamp mee; InfluxDB zal die automatisch toevoegen. Er moet een spatie staan tussen tags en fields !



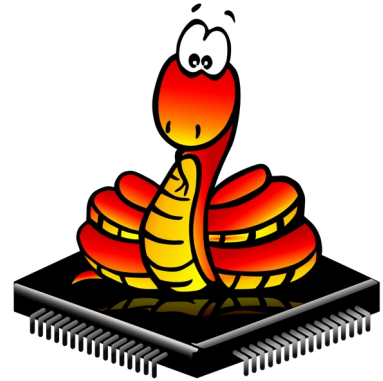
# Software

## Meetstation

[MicroPython](#) is een implementatie van Python 3, geschreven in C en geoptimaliseerd om op microcontrollers te werken. Net zoals bij Python wordt de code niet gecompileerd, maar tijdens het uitvoeren geïnterpreteerd.

De sturing van het meetstation bevat volgende functionaliteit :

- verbinden met het lokale WiFi netwerk
- weer info ophalen van OpenWeatherMap.org
- sensoren en batterijspanning uitlezen
- meetwaarden doorsturen naar de MQTT broker op de RPi4 backend
- microcontroller in deepsleep brengen tot de volgende meting



## MicroPython firmware

### Download firmware

De meest recente versie van MicroPython downloaden we [hier](#). De installatie instructies staan op dezelfde pagina. We gebruiken hiervoor de *esptool.py* tool die je van de [github van Espressif](#) kan halen.

### Wissen ESP32 flash

```
esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
```

### Uploaden MicroPython naar ESP32 flash

```
esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash -z 0x1000 esp32-20220117-v1.18.bin
```

### Testen MicroPython op ESP32

```
picocom /dev/ttyUSB0 -b115200
```

Druk een paar keer op Enter en we krijgen de REPL prompt.

```
Terminal ready
>>> print('Hello ESP32!')
Hello ESP32!
>>>
```



## MicroPython code

### Modules

Module	Beschrijving	Installatie
machine	Hardware functies	(ingebouwd)
network	netwerk configuratie	(ingebouwd)
json	JSON encoding/decoding	(ingebouwd)
umqtt.simple	MQTT functies	(ingebouwd)
urequests	HTTP bibliotheek	(ingebouwd)
sys	Systeem functies	(ingebouwd)
utime	Tijd gerelateerde functies	(ingebouwd)
am2320	Aosong AM2320 I2C driver	<a href="https://github.com/mcauser/micropython-am2320">https://github.com/mcauser/micropython-am2320</a>
bmp180	Bosch BMP180 I2C driver	<a href="https://github.com/micropython-IMU/micropython-bmp180">https://github.com/micropython-IMU/micropython-bmp180</a>
bh1750	Mouser Electronics BH1750 I2C driver	<a href="https://github.com/PinkInk/upylib/tree/master/bh1750">https://github.com/PinkInk/upylib/tree/master/bh1750</a>

### Configuratie

Module	Variabele	Beschrijving
config	SCL_PIN	I2C clock Pin
config	SDA_PIN	I2C data Pin
config	LED_PIN	Onboard LED Pin gebruikt als fout indicator
config	LED_ON & LED_OFF	Omgekeerde logica van onboard LED
config	DEBUG_PIN	Pin voor debugging (LOW is debug AAN)
config	VBAT_PIN	Pin voor batterijspanning monitoring
config	FAHRENHEIT	Temperatuur in Fahrenheit (True/False)
config	INTERVAL	Interval tussen sensor metingen (seconden)
config	SSID & PASS	SSID en paswoord voor het WiFi netwerk
config	MAX_TRIES	Maximum aantal pogingen voor connectie op het WiFi netwerk (1 sec interval)
config	MQTT_HOST	MQTT host voor uploaden sensor metingen

Module	Variabele	Beschrijving
config	MQTT_TOPIC	MQTT topic voor uploaden sensor metingen
config	OPENWEATHERMAP_API	OpenWeather API sleutel
config	OPENWEATHERMAP_CITY	OpenWeather stad,land voor ophalen weer info
config	OPENWEATHERMAP_LAT	OpenWeather latitude voor ophalen weer info
config	OPENWEATHERMAP_LON	OpenWeather longitude voor ophalen weer info
config	OPENWEATHERMAP_URL	OpenWeather url voor huidige weer info

## Funcities

Functie	Beschrijving
show_error()	Fout conditie laat de onboard LED knipperen
debug_on()	Controle debug pin LOW (debug AAN)
connect_wifi()	Verbinden µcontroller met het WiFi netwerk
get_weather_data()	Ophalen van de huidige weer info van OpenWeatherMap.org
temperature_2_unit()	Geef de temperatuur weer in de juiste eenheid ( config.FAHRENHEIT)
get_sensor_readings()	Lees de waarden van de sensoren uit
log_readings()	Stuur de metingen via MQTT naar de backend
deepsleep_till_next_cycle()	Breng de µcontroller in deepsleep voor config.INTERVAL seconden
run()	Programma logica

De volledige source code kan je hier vinden.

## Backend

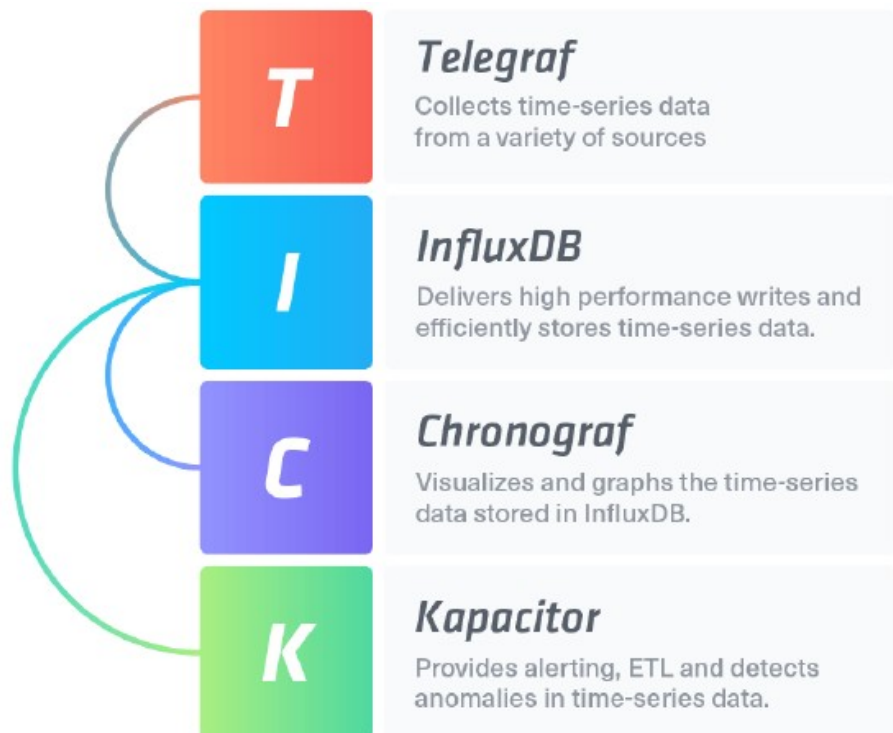
De backend draait op een Raspberry Pi 4B 2GB en het Raspberry Pi Bullseye 64bit OS. Voor het verwerken van de meetgegevens zullen we gebruik maken van volgende [docker](#) images:

- **Mosquitto** : MQTT broker voor het ontvangen van de meetwaarden van het meetstation.
- **Telegraf** : doorsturen van de MQTT meetwaarden naar de database
- **InfluxDB** : tijd gerelateerde database voor het opslaan van de meetwaarden, het visualiseren en monitoren ervan



In InfluxDB 2.x zijn de database (*InfluxDB*), de visualisatie (*Chronograf*) en monitoring (*Kapacitor*) geïntegreerd in één pakket en vormen ze te samen met *Telegraf* de zogenaamde [TICK stack](#) die een mooie oplossing biedt voor onze noden.

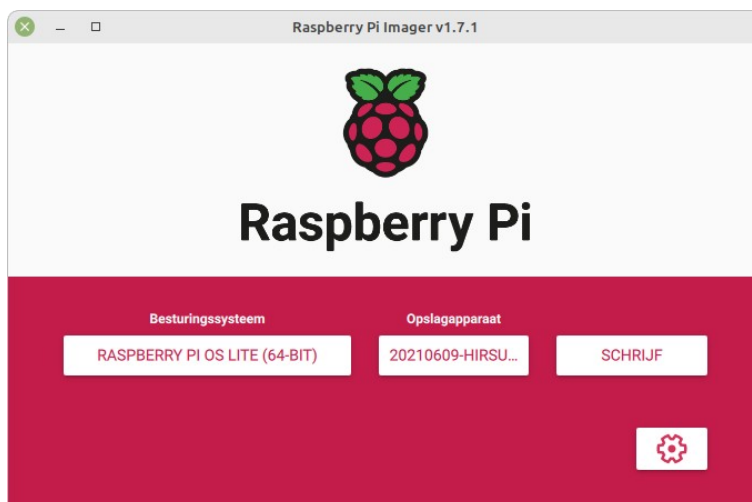
Als je kiest voor InfluxDB 1.x kan je hetzelfde opzetten maar dan moet je wel Chronograf en Kapacitor zelf installeren en mis je de geïntegreerde web UI van versie 2.x



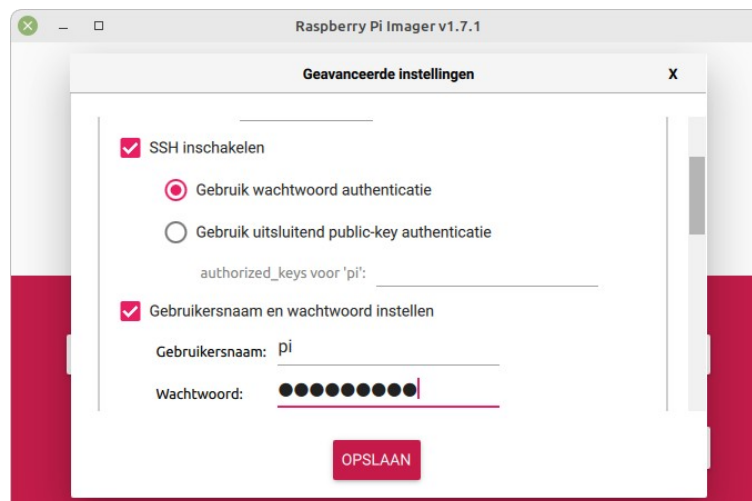
We zullen deze docker containers configureren in een YAML file en besturen met [docker-compose](#).

## Raspberry Pi OS

We installeren het *Raspberry Pi Bullseye Lite 64bit OS* met behulp van Imager op een SanDisk 16GB USB flash drive.



We klikken op het configuratie icoontje en schakelen SSH aan zodat we op de Pi kunnen inloggen via het netwerk.



Na de nodige bevestigingen wordt het OS op de USB flash drive geschreven en geverifieerd.



Daarna plaatsen we de USB flash drive in een USB 3 poort op de Pi, sluiten de netwerkkabel aan en vervolgens de power adapter. Nadat de Rpi is opgestart loggen we in via SSH.

We werken het OS bij met :

***sudo apt update***

***sudo apt upgrade -y***

```
pi@piweather:~$ sudo apt update
Hit:1 http://deb.debian.org/debian bullseye InRelease
Get:2 http://deb.debian.org/debian bullseye-updates InRelease [39.4 kB]
Get:3 http://security.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:4 http://archive.raspberrypi.org/debian bullseye InRelease [23.6 kB]
Get:5 http://security.debian.org/debian-security bullseye-security/main arm64 Packages [121 kB]
Get:6 http://security.debian.org/debian-security bullseye-security/main armhf Packages [124 kB]
Get:7 http://security.debian.org/debian-security bullseye-security/main Translation-en [77.5 kB]
Get:8 http://archive.raspberrypi.org/debian bullseye/main arm64 Packages [260 kB]
Get:9 http://archive.raspberrypi.org/debian bullseye/main armhf Packages [264 kB]
Fetched 953 kB in 2s (622 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
18 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi@piweather:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  bind9-host bind9-libs libcamera-apps-lite libcamera0 libcryptsetup12 libexpat1 libssl2-2 libssl2-modules
  libssl2-modules-db libssl1.1 libwbclient0 linux-libc-dev openssl raspberrypi-bootloader raspberrypi-kernel
  raspberrypi-sys-mods raspi-config rpi-eeeprom
10 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 35.1 MB of archives.
After this operation, 3,747 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.raspberrypi.org/debian bullseye/main arm64 libssl1.1 arm64 1.1.1k-1+deb11u2+rpt1 [1,390 kB]
Get:2 http://security.debian.org/debian-security bullseye-security/main arm64 libwbclient0 arm64 2:4.13.13+dfsg-1-deb1
  1u3 [307 kB]
Get:3 http://security.debian.org/debian-security bullseye-security/main arm64 bind9-libs arm64 1:9.16.27-1-deb11u1 [1,
  297 kB]
Get:4 http://security.debian.org/debian-security bullseye-security/main arm64 bind9-host arm64 1:9.16.27-1-deb11u1 [30
  1 kB]
Get:5 http://security.debian.org/debian-security bullseye-security/main arm64 libcryptsetup12 arm64 2:2.3.7-1-deb11u1
  [230 kB]
Get:6 http://security.debian.org/debian-security bullseye-security/main arm64 libexpat1 arm64 2.2.10-2+deb11u3 [84.1 k
  B]
Get:7 http://security.debian.org/debian-security bullseye-security/main arm64 libssl2-modules-db arm64 2.1.27+dfsg-2.
  1+deb11u1 [69.4 kB]
Get:8 http://security.debian.org/debian-security bullseye-security/main arm64 libssl2-2 arm64 2.1.27+dfsg-2.1+deb11u1
  [105 kB]
Get:9 http://security.debian.org/debian-security bullseye-security/main arm64 libssl2-modules arm64 2.1.27+dfsg-2.1+d
  eb11u1 [101 kB]
Get:10 http://archive.raspberrypi.org/debian bullseye/main arm64 raspberrypi-kernel arm64 1:1.20220308-2 [21.9 MB]
Get:11 http://archive.raspberrypi.org/debian bullseye/main arm64 libcamera0 arm64 0-g1t20220303+e68e0f1e-1 [756 kB]
Get:12 http://archive.raspberrypi.org/debian bullseye/main arm64 libcamera-apps-lite arm64 0-g1t20220228+0dc5ea8-1 [25
  9 kB]
Get:13 http://archive.raspberrypi.org/debian bullseye/main arm64 linux-libc-dev arm64 1:1.20220308-2 [1,023 kB]
Get:14 http://archive.raspberrypi.org/debian bullseye/main arm64 openssl arm64 1.1.1k-1+deb11u2+rpt1 [829 kB]
Get:15 http://archive.raspberrypi.org/debian bullseye/main arm64 raspberrypi-bootloader arm64 1:1.20220308-2 [4,527 kB]
81% [15 raspberrypi-bootloader 14.0 kB/4,527 kB 0%]
```

# Docker

We installeren docker :

**curl -sSL https://get.docker.com | sh**

Daarna herstarten we de Pi en controleren of de docker service draait :

**systemctl status docker.service**

Standaard kan docker alleen uitgevoerd worden met root rechten en moeten we *sudo* gebruiken. Om dit met gebruiker pi te kunnen doen :

**sudo usermod -aG docker pi**  
**newgrp docker**

```
pi@rpiweather: ~  
Bestand  Bewerken  Beeld  Zoeken  Terminal  Hulp  
+ sudo -E sh -c apt-get update -qq >/dev/null  
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends docker-ce-cli docker-ce  
>/dev/null  
+ version gte 20.10  
+ [ -z ]  
+ return 0  
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras >/dev/null  
+ sudo -E sh -c docker version  
Client: Docker Engine - Community  
Version: 20.10.13  
API version: 1.41  
Go version: go1.16.15  
Git commit: a224086  
Built: Thu Mar 10 14:07:19 2022  
OS/Arch: linux/arm64  
Context: default  
Experimental: true  
  
Server: Docker Engine - Community  
Engine:  
Version: 20.10.13  
API version: 1.41 (minimum version 1.12)  
Go version: go1.16.15  
Git commit: 906f57f  
Built: Thu Mar 10 14:05:37 2022  
OS/Arch: linux/arm64  
Experimental: false  
containerd:  
Version: 1.5.10  
GitCommit: 2a1d4ddb2a1030dc5b01e96fb110a9d9f150ecc  
runc:  
Version: 1.0.3  
GitCommit: v1.0.3-0-gf46b6ba  
docker-init:  
Version: 0.19.0  
GitCommit: de40ad0  
  
=====
```

To run Docker as a non-privileged user, consider setting up the Docker daemon in rootless mode for your user:

```
dockerd-rootless-setupool.sh install
```

Visit <https://docs.docker.com/go/rootless/> to learn about rootless mode.

To run the Docker daemon as a fully privileged service, but granting non-root users access, refer to <https://docs.docker.com/go/daemon-access/>

WARNING: Access to the remote API on a privileged Docker daemon is equivalent to root access on the host. Refer to the 'Docker daemon attack surface' documentation for details: <https://docs.docker.com/go/attack-surface/>

```
=====
```

pi@rpiweather:~\$

We testen of docker correct werkt :

**docker version**  
**docker run hello-world**

```
pi@rpiweather: ~  
Bestand  Bewerken  Beeld  Zoeken  Terminal  Hulp  
pi@rpiweather:~$ docker version  
Client: Docker Engine - Community  
Version: 20.10.13  
API version: 1.41  
Go version: go1.16.15  
Git commit: a224086  
Built: Thu Mar 10 14:07:19 2022  
OS/Arch: linux/arm64  
Context: default  
Experimental: true  
  
Server: Docker Engine - Community  
Engine:  
Version: 20.10.13  
API version: 1.41 (minimum version 1.12)  
Go version: go1.16.15  
Git commit: 906f57f  
Built: Thu Mar 10 14:05:37 2022  
OS/Arch: linux/arm64  
Experimental: false  
containerd:  
Version: 1.5.10  
GitCommit: 2a1d4ddb2a1030dc5b01e96fb110a9d9f150ecc  
runc:  
Version: 1.0.3  
GitCommit: v1.0.3-0-gf46b6ba  
docker-init:  
Version: 0.19.0  
GitCommit: de40ad0  
  
pi@rpiweather:~$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
7050e35b49f5: Pull complete  
Digest: sha256:bfeaa0278a0a267fad2634554f4f0c6f31981eea41c553fd5a83e95a41d40c38  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (arm64v8)  
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
  
pi@rpiweather:~$
```

## Docker-Compose

Dit moet met pip3 geïnstalleerd worden.

**sudo apt install python3-pip**  
**sudo pip3 install docker-compose**

Alle gebruikte images zullen in *docker-compose.yml* worden geconfigureerd zodat ze met één docker-compose commando gezamenlijk kunnen worden bestuurd.

We zetten deze folderstructuur op :

```
├── docker
│   ├── mosquito
│   └── telegraf
```

```
pi@rpiweather: ~
Bestand Bewerken Beeld Zoeken Terminal Hulp

Collecting dockerpty<1,>=0.4.1
  Downloading https://www.piwheels.org/simple/dockerpty/dockerpty-0.4.1-py3-none-any.whl (16 kB)
Collecting jsonschema<4,>=2.5.1
  Downloading https://www.piwheels.org/simple/jsonschema/jsonschema-3.2.0-py2.py3-none-any.whl (56 kB)
  56 kB 2.2 MB/s
Collecting docopt<1,>=0.6.1
  Downloading https://www.piwheels.org/simple/docopt/docopt-0.6.2-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: requests<3,>=2.20.0 in /usr/lib/python3/dist-packages (from docker-compose) (2.25.1)
Collecting paramiko>=2.4.2
  Downloading https://www.piwheels.org/simple/paramiko/paramiko-2.10.3-py2.py3-none-any.whl (214 kB)
  214 kB 3.1 MB/s
Requirement already satisfied: six>=1.3.0 in /usr/lib/python3/dist-packages (from dockerpty<1,>=0.4.1->docker-compose) (1.16.0)
Collecting pyrsistent>=0.14.0
  Downloading pyrsistent-0.18.1.tar.gz (100 kB)
  100 kB 4.3 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing wheel metadata ... done
Collecting attrs>=17.4.0
  Downloading https://www.piwheels.org/simple/attrs/attrs-21.4.0-py2.py3-none-any.whl (60 kB)
  60 kB 923 kB/s
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from jsonschema<4,>=2.5.1->docker-compose) (52.0.0)
Collecting cryptography>=2.5
  Downloading cryptography-36.0.2-cp36-abi3-manylinux_2_17_aarch64.manylinux2014_aarch64.manylinux_2_24_aarch64.whl (3.3 MB)
  3.3 MB 2.0 MB/s
Collecting bcrypt>=3.1.3
  Downloading bcrypt-3.2.0-cp36-abi3-manylinux2014_aarch64.whl (56 kB)
  56 kB 2.2 MB/s
Collecting pynacl>=1.0.1
  Downloading PyNaCl-1.5.0-cp36-abi3-manylinux_2_17_aarch64.manylinux2014_aarch64.manylinux_2_24_aarch64.whl (601 kB)
  601 kB 11.8 MB/s
Collecting cffi>=1.1
  Downloading cffi-1.15.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (447 kB)
  447 kB 14.4 MB/s
Collecting pycparser
  Downloading https://www.piwheels.org/simple/pycparser/pycparser-2.21-py2.py3-none-any.whl (119 kB)
  119 kB 1.2 MB/s
Building wheels for collected packages: pyrsistent
  Building wheel for pyrsistent (PEP 517) ... done
  Created wheel for pyrsistent: filename=pyrsistent-0.18.1-cp39-cp39-linux_aarch64.whl size=106123 sha256=1e8ea73c524e405b70f8a9c66354804dcb1ae0121629322fb0f9a28ea998c439
  Stored in directory: /root/.cache/pip/wheels/87/fe/e6/fc8deeb581a41e462eafaf19fee96f51cdc8391e0be1c8088a
Successfully built pyrsistent
Installing collected packages: pycparser, cffi, websocket-client, pynacl, cryptography, bcrypt, pyrsistent, paramiko, docker, attrs, texttable, PyYAML, python-dotenv, jsonschema, docopt, dockerpty, docker-compose
Successfully installed PyYAML-5.4.1 attrs-21.4.0 bcrypt-3.2.0 cffi-1.15.0 cryptography-36.0.2 docker-5.0.3 docker-comp
ose-1.29.2 dockerpty-0.4.1 docopt-0.6.2 jsonschema-3.2.0 paramiko-2.10.3 pycparser-2.21 pynacl-1.5.0 pyrsistent-0.18.1
python-dotenv-0.19.2 texttable-1.6.4 websocket-client-0.59.0
pi@rpiweather:~$ docker-compose version
docker-compose version 1.29.2, build unknown
docker-py version: 5.0.3
CPython version: 3.9.2
OpenSSL version: OpenSSL 1.1.1.k 25 Mar 2021
pi@rpiweather:~$
```

## Docker network

Omdat alle docker containers data met elkaar moeten kunnen uitwisselen laten we ze draaien in hun eigen netwerk *iot*. Hiervoor kunnen ze elkaar aanroepen op naam ipv op IP adres.

**docker network create iot**  
**docker network ls**

```
pi@rpiweather: ~
Bestand Bewerken Beeld Zoeken Terminal Hulp

pi@rpiweather:~$ docker network create iot
002c70b891103141d40d990a0dab26decafe5f23ce1b1549ef601fea007250
pi@rpiweather:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
84330e40bc3f        bridge              bridge              local
e2ac2f36e6e3        docker_default      bridge              local
f3ff4c5fce42        host                host                local
002c70b89110        iot                 bridge              local
498e9814c524        none                null                local
pi@rpiweather:~$
```

## Docker containers

### *Eclipse Mosquitto*

In deze image zijn folders voorzien voor configuratie, data en logging. We zullen de default waarden voor deze parameters instellen via een lokaal configuratie bestand *mosquitto.conf*.

```
listener 1883
allow_anonymous true
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

*mosquitto.conf*

Vanaf versie 2.0 van mosquitto moet je een authenticatie methode configureren voor clients. Bij vroegere versies konden clients zonder gebruikersnaam en paswoord verbinden met de server. We stellen hier in dat clients op poort 1883 toch nog anoniem kunnen verbinden.

De rest van de configuratie heeft te maken met de persistentie van de gegevens en de logs na het herstarten van de container. Standaard begint de container bij een (her)start terug met een schone lei, d.w.z. dat alle gegevens van de vorige run verloren zijn gegaan. In vele gevallen is dit niet wenselijk en moeten we maatregelen treffen om de gegevens te bewaren.

Dit doen we door zogenaamde **named volumes** te configureren. De gegevens van de container worden gemapt naar deze named volumes op de host rpi. De named volumes bevinden zich in de folder */var/lib/docker/volumes* op de raspberry pi.

Het docker-compose bestand voor deze image bevat instructies hoe de image moet worden gestart.

```
version: "3.8"

services:
  mosquitto:
    container_name: mosquitto
    image: eclipse-mosquitto:2.0
    ports:
      - "1883:1883"
      - "9001:9001"
    networks:
      - iot
    volumes:
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
      - mosquitto-data:/mosquitto/data
      - mosquitto-log:/mosquitto/log

networks:
  iot:

volumes:
  mosquitto-data:
  mosquitto-log:
```

*docker-compose.yml (mosquitto)*

We kunnen nu de image met één eenvoudig commando starten :

**docker-compose up**



## Influxdb

De database is vanaf de versie 2.x ingrijpend veranderd t.o.v. de 1.x in de manier waarop ze geconfigureerd moet worden. Gelukkig bestaat er een officiële image die veel van de configuratie automatiseert. We kunnen een *set-up* configureren die automatisch een gebruiker, wachtwoord, organisatie, bucket (=database) en token aanmaakt. De parameters voor deze set-up worden via *omgevingsvariabelen* doorgegeven.

We configureren uiteraard **named volumes** om de data en configuratie persistent te maken.

```
version: "3.8"

services:
  mosquitto:
    ...
    ...

  influxdb:
    container_name: influxdb
    image: influxdb:2.2
    ports:
      - "8086:8086"
    networks:
      - iot
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=${DOCKER_INFLUXDB_INIT_USERNAME}
      - DOCKER_INFLUXDB_INIT_PASSWORD=${DOCKER_INFLUXDB_INIT_PASSWORD}
      - DOCKER_INFLUXDB_INIT_ORG=effevee
      - DOCKER_INFLUXDB_INIT_BUCKET=weatherdata
      - DOCKER_INFLUXDB_INIT_RETENTION=365d
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}
    volumes:
      - influxdb-config:/etc/influxdb2
      - influxdb-data:/var/lib/influxdb2

networks:
  iot:

volumes:
  mosquitto-data:
  mosquitto-log:
  influxdb-config:
  influxdb-data:
```

*docker-compose.yaml (influxdb)*

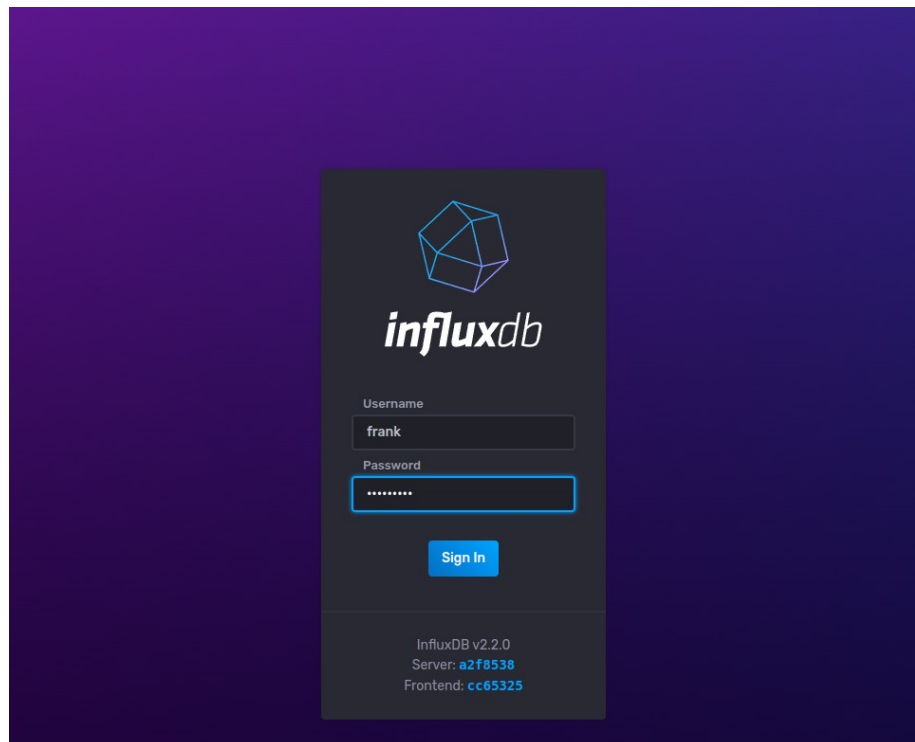
Voor de veiligheid plaatsen we alle gevoelige informatie zoals paswoorden en tokens in een *.env* bestand in onze docker directory zodat we in de *docker-compose.yaml* deze kunnen aanroepen met ***\${variabele}***

```
DOCKER_INFLUXDB_INIT_USERNAME=<username>
DOCKER_INFLUXDB_INIT_PASSWORD=<password>
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=<admin_token>
```

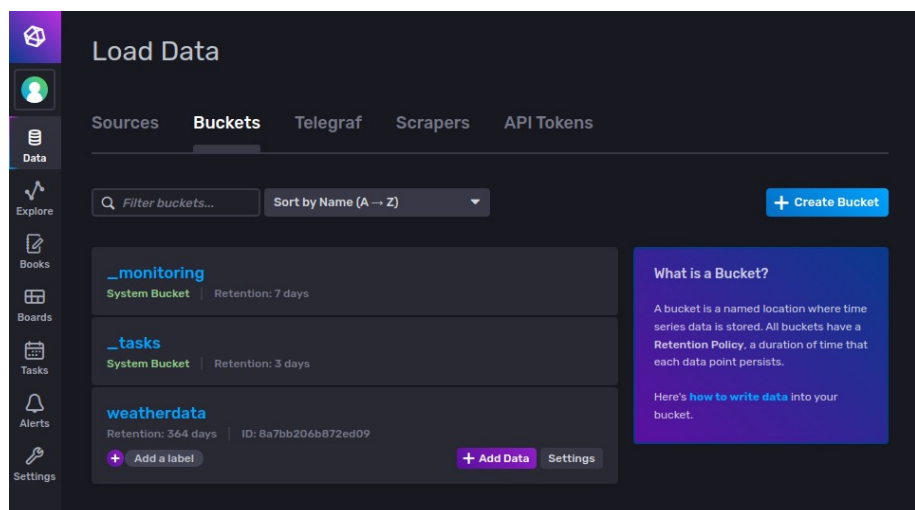
*.env (influxdb)*

We starten de images nu op met het gekende **docker-compose up** commando.

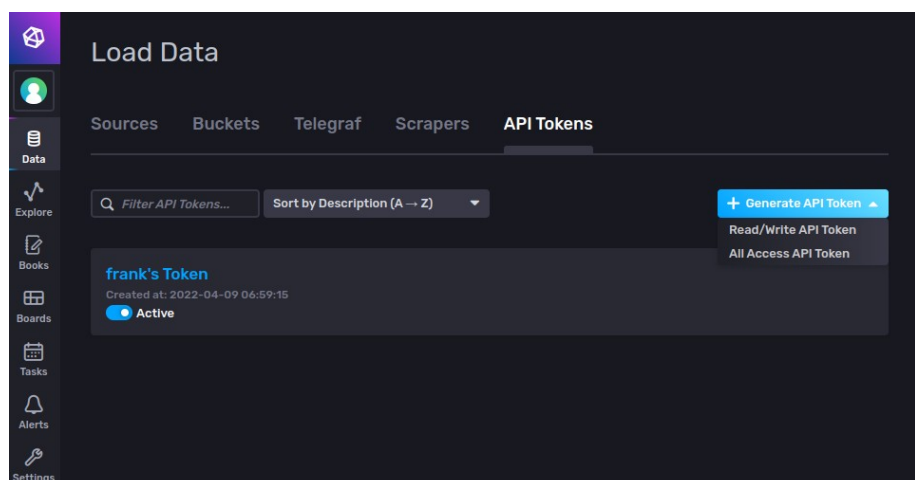
Als alles opstart zonder fouten kunnen we daarna inloggen op de vernieuwde influxdb web UI  
`http://rpiweather.local:8086`  
met de aangemaakte **gebruiker** en **wachtwoord**.



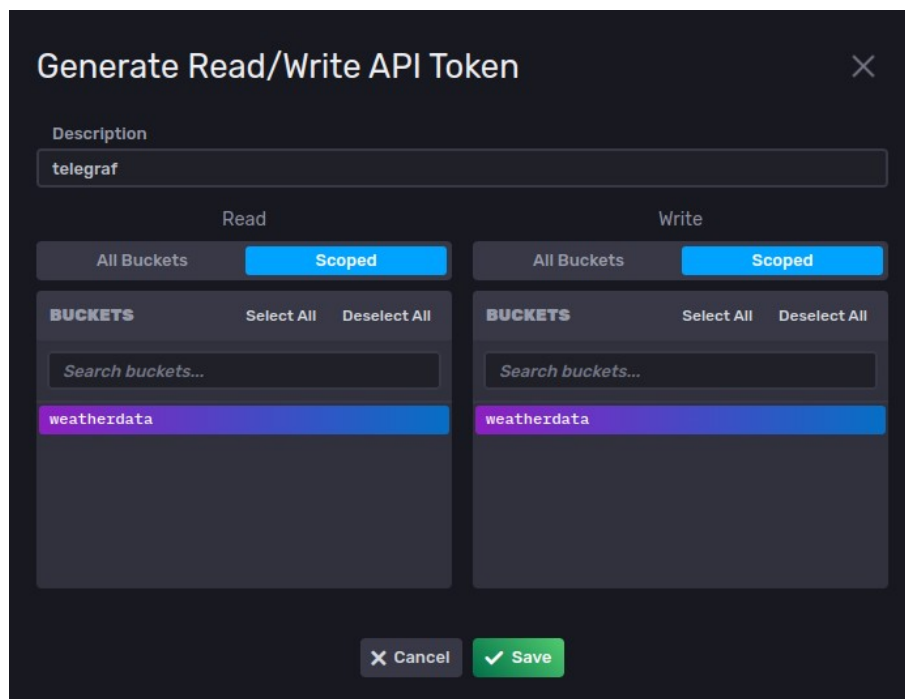
We selecteren daarna *Data – Buckets* en zien dat onze database **weatherdata** inderdaad is aangemaakt. Daar we bij de setup een *retention policy 52w* hebben meegegeven zal de data die ouder is dan 52 weken (1 jaar) automatisch verwijderd worden.



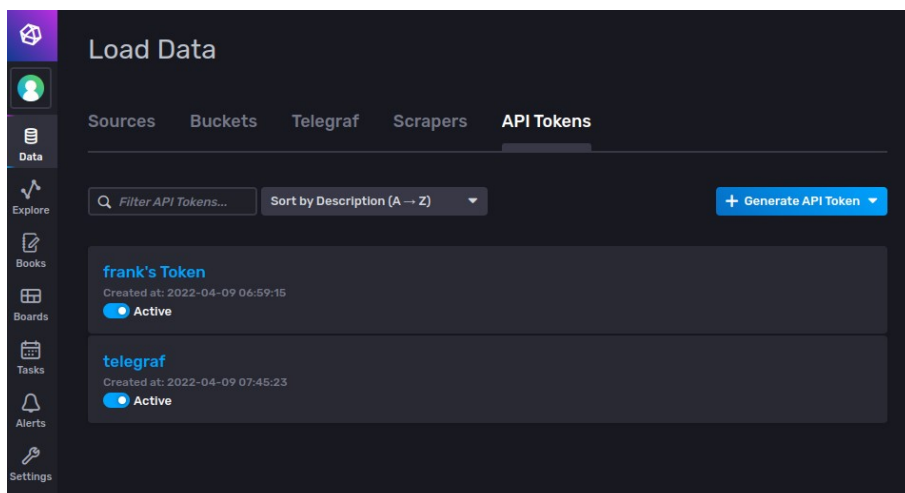
We moeten nu nog een Token aanmaken voor telegraf met lees/schrijf rechten op de database. Klik op *Data – API Tokens* en vervolgens op de knop *Generate API Token – Read/Write API Token*.



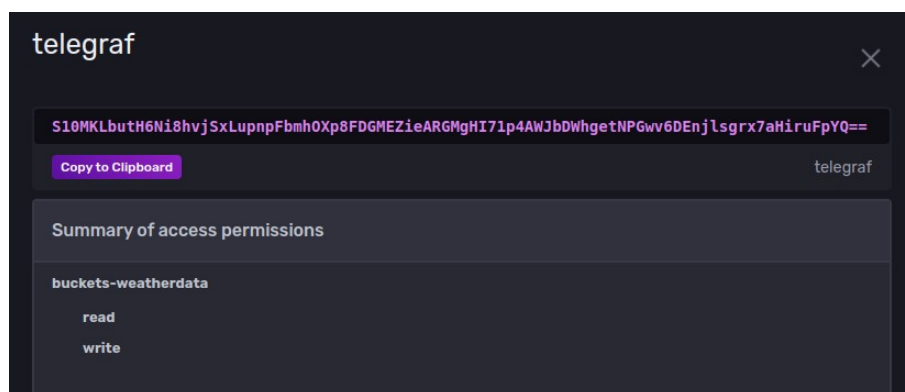
Vul in *Description* **telegraf**, selecteer de bucket **weatherdata** in *Read* en *Write* en klik op *Save*.



De Token voor telegraf is nu aangemaakt. Klik erop.



Klik op *Copy to Clipboard* en bewaar de Token. Je zal die nodig hebben in de volgende stap tijdens de configuratie van telegraf.



## Telegraf

Telegraf werkt volledig via plugins. We zullen een lokaal configuratie bestand *telegraf.conf* gebruiken om alles in te stellen. Het blokje **[agent]** bevat algemene instellingen. De input van de data zit in **[[inputs.mqtt\_consumer]]**. Deze definieert de MQTT server en de topic waarnaar moet geluisterd worden. **[[outputs.influxdb\_v2]]** bepaalt de bestemming van de data : de influxdb server en de nodige informatie die nodig is om te kunnen verbinden en de data te kunnen schrijven in de database.

```
[agent]
  interval = "10s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  collection_jitter = "0s"
  flush_interval = "10s"
  flush_jitter = "0s"
  precision = ""
  debug = false
  quiet = false
  logfile = ""
  hostname = ""
  omit_hostname = false

[[outputs.influxdb_v2]]
  urls = ["http://influxdb:8086"]
  token = "$DOCKER_INFLUXDB_TELEGRAF_TOKEN"
  organization = "$DOCKER_INFLUXDB_INIT_ORG"
  bucket = "$DOCKER_INFLUXDB_INIT_BUCKET"
  insecure_skip_verify = true

[[outputs.file]]
  files = ["stdout", "/tmp/weatherdata.out"]

[[inputs.mqtt_consumer]]
  servers = ["tcp://mosquitto:1883"]
  topics = [
    "weatherdata"
  ]
  data_format = "influx"
```

*telegraf.conf*

We configureren de image terug in het docker-compose bestand. We moeten hier in ieder geval de 3 **omgevingsvariabelen** definiëren die in het configuratie bestand te zien zijn, oa ook de token die we in vorige stap hebben gemaakt voor telegraf. We moeten er ook voor zorgen dat het lokale configuratie bestand gebruikt wordt. We laten de image ook pas opstarten als mosquitto en influxdb reeds operationeel zijn omdat telegraf beiden nodig heeft om zijn werk te kunnen doen. Er zijn geen named volumes gedefinieerd.

```

version: "3.8"

services:
  mosquitto:
    ...
    ...

  influxdb:
    ...
    ...

  telegraf:
    container_name: telegraf
    image: telegraf:1.22
    networks:
      - iot
    environment:
      - DOCKER_INFLUXDB_INIT_ORG=effevee
      - DOCKER_INFLUXDB_INIT_BUCKET=weatherdata
      - DOCKER_INFLUXDB_TELEGRAF_TOKEN=${DOCKER_INFLUXDB_TELEGRAF_TOKEN}
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
    depends_on:
      - mosquitto
      - influxdb

networks:
  iot:

volumes:
  mosquitto-data:
  mosquitto-log:
  influxdb-config:
  influxdb-data:

```

*docker-compose.yaml (telegraf)*

Het in vorige stap aangemaakte telegraf token voegen we ook toe aan ons *.env* bestand zodat we in de yaml file geen gevoelige informatie moeten plaatsen.

```

...
...
DOCKER_INFLUXDB_TELEGRAF_TOKEN=<telegraf_token>

```

*.env (telegraf)*

We starten de images nu op met het gekende **docker-compose up** commando.

# InfluxDB dashboard

De grafieken van het dashboard maken we aan in de vernieuwde web UI van InfluxDB 2.x. De interface is heel makkelijk in gebruik en je hebt in no time een mooie voorstelling van je gegevens. Het voordeel om deze interface te gebruiken in plaats van grafana is dat je daar de nieuwe flux query taal moet gebruiken die wel wat wennen is. Bij InfluxDB kun je interactief de grafiek configureren en wordt de flux query in de achtergrond automatisch aangemaakt.

Er zijn heel wat verschillende [grafiek types](#) :

- Band
- Gauge
- Graph
- Graph + Single Stat
- Heatmap
- Histogram
- Mosaic
- Scatter
- Single Stat
- Table

De grafieken kun je makkelijk in een overzichtelijk dashboard schikken.



## Temperatuur

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals" or r["_measurement"] ==
"forecasts")
  |> filter(fn: (r) => r["_field"] == "ac_temp" or r["_field"] == "ow_temp")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

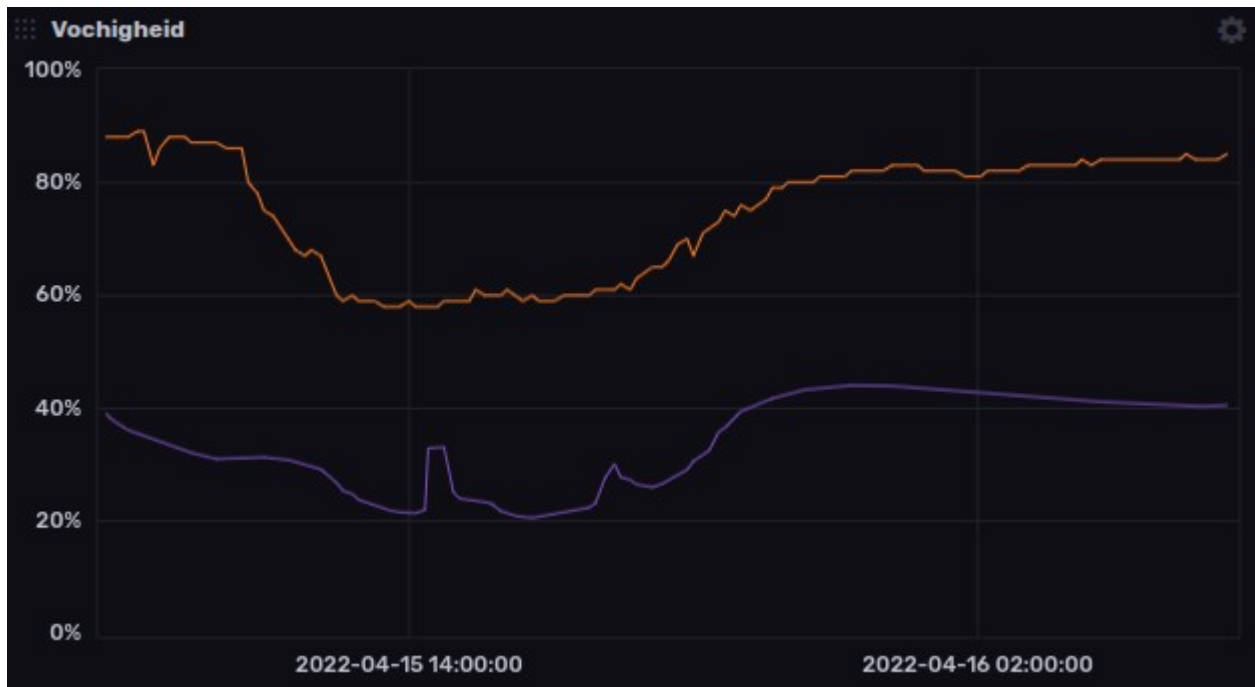
De grafiek is van het type *Graph*.



## Vochtigheid

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals" or r["_measurement"] ==
"forecasts")
  |> filter(fn: (r) => r["_field"] == "ac_hum" or r["_field"] == "ow_hum")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

De grafiek is van het type *Graph*.





# Luchtdruk

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals" or r["_measurement"] ==
"forecasts")
  |> filter(fn: (r) => r["_field"] == "ac_pres" or r["_field"] == "ow_pres")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

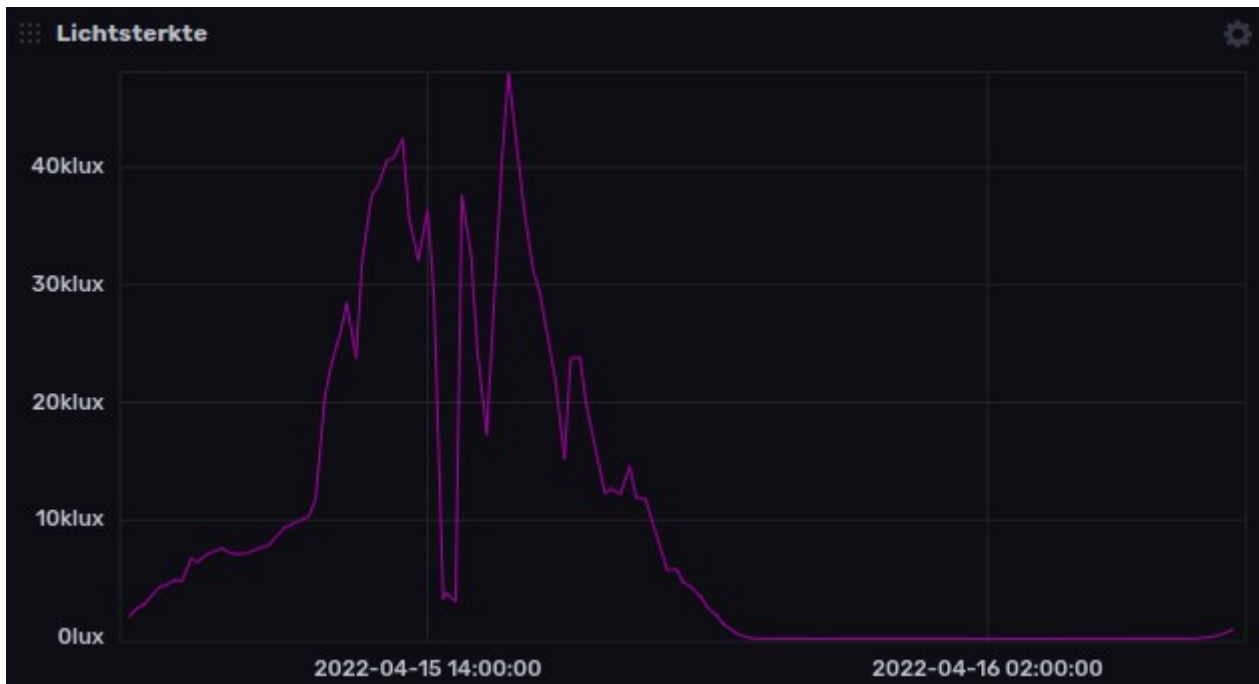
De grafiek is van het type *Graph*.



## Lichtsterkte

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals" or r["_measurement"] ==
"forecasts")
  |> filter(fn: (r) => r["_field"] == "ac_lum")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

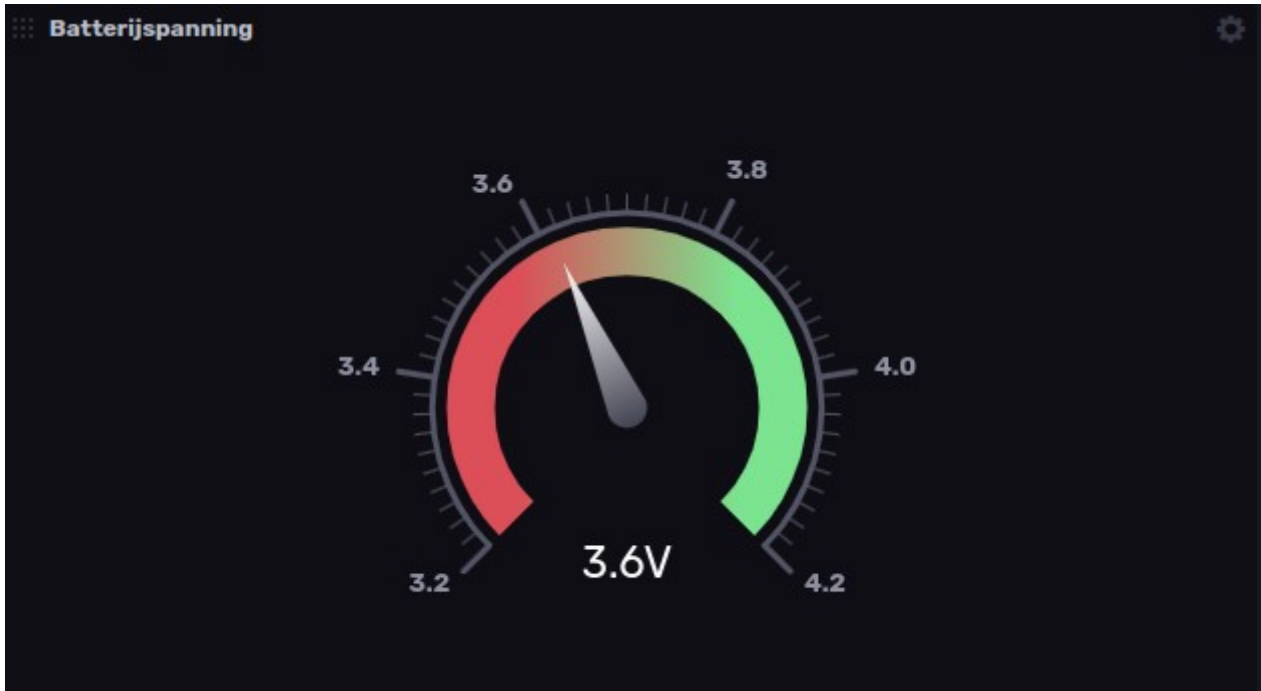
De grafiek is van het type *Graph*.



## Batterijspanning

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals")
  |> filter(fn: (r) => r["_field"] == "ac_batv")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

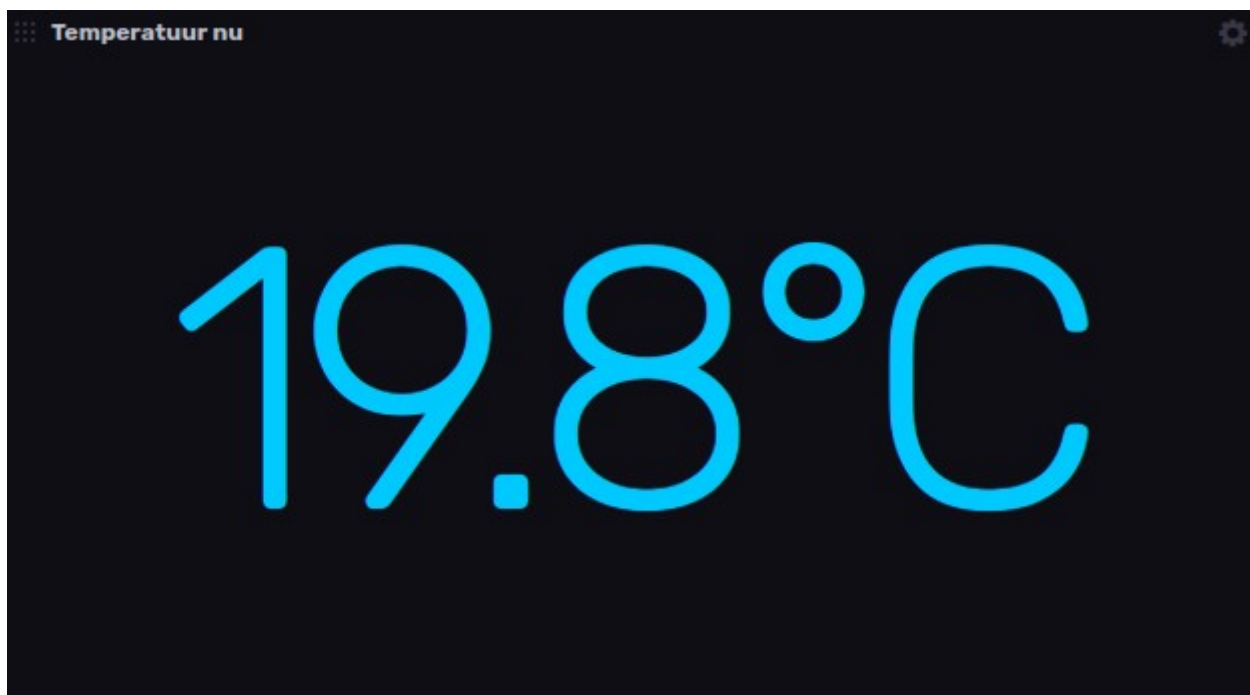
De grafiek is van het type *Gauge*.



## Temperatuur nu

```
from(bucket: "weatherdata")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "actuals")
  |> filter(fn: (r) => r["_field"] == "ac_temp")
  |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
  |> yield(name: "last")
```

De grafiek is van het type *Single Stat*.



# InfluxDB monitoring

De vernieuwde UI web interface van InfluxDB2 heeft de mogelijkheid om te controleren als bepaalde waarden overschreden worden (*threshold checks*) of niet meer binnenkomen (*deadman checks*). De statussen van deze checks (CRIT, WARN, INFO of OK) worden op hun beurt opgeslagen in de systeem bucket `_monitoring`. Aan de hand van de gegevens in deze tabellen kunnen we vervolgens waarschuwingen genereren die we bijvoorbeeld via email kunnen versturen.

In ons geval zullen de volgende parameters gemonitord worden :

- **controle batterijspanning** : het meetstation wordt gevoed met een lithium 18650 batterij die via een zonnepaneeltje en een TP4056 wordt bijgeladen. Een LDO HT7333 spanningsregelaar zorgt voor een constante spanning van 3.3V voor de voeding van de ESP32 en de sensoren. Het is dus belangrijk dat de spanning van de batterij boven de 3.4V blijft voor de goede werking. Hiervoor zullen we een *threshold check* aanmaken.
- **controle meetstation** : het niet meer doorsturen van gegevens kan verschillende oorzaken hebben: ESP32 kan geen Wifi connectie maken, voeding meetstation defect, ESP32 gecrasht, ... Hiervoor zullen we een *deadman check* aanmaken.

We beginnen met het aanmaken van de controle checks. De documentatie vind je [hier](#).

## Controle batterijspanning

We klikken op *Alerts* en vervolgens op *Create – Threshold Check*. We selecteren **1. Define Query**. We kiezen het veld `ac_batv` uit de tabel *actuals* van de *weatherdata* database. Enkel de laatste waarde (*last*) wordt geselecteerd. Klik op *Submit*

The screenshot shows the InfluxDB2 web interface for creating a threshold check. The title is 'Controlle batterijspanning'. There are two tabs: '1. Define Query' (active) and '2. Configure Check'. Below the tabs is a large empty box for a visualization. At the bottom, there is a query editor with the following sections:

- FROM:** A search bar 'Search for a bucket' and a list of buckets: `_monitoring`, `_tasks`, and `weatherdata` (selected).
- Filter:** A dropdown for 'Filter' with a search bar 'Search \_measurement tag values'. The selected filter is 'actuals'.
- Field:** A dropdown for 'Filter' with a search bar 'Search \_field tag values'. The selected field is 'ac\_batv'.
- Host:** A dropdown for 'Filter' with a search bar 'Search host tag values'. The selected host is '47c2dd3def0b'.
- WINDOW PERIOD:** A dropdown with options: 'auto (10m)', 'skew', 'spread', 'stddev', 'first', 'last' (selected), 'unique', and 'sort'.
- AGGREGATE FUNCTION:** A dropdown with options: 'skew', 'spread', 'stddev', 'first', 'last' (selected), 'unique', and 'sort'.
- Submit:** A blue button to submit the query.

Vervolgens klikken we op **2. Configure Check**. We laten de check iedere 10m uitvoeren, dit is ook de frequentie waarmee de metingen worden gedaan. Daarnaast definiëren we ook de volgende *thresholds* :

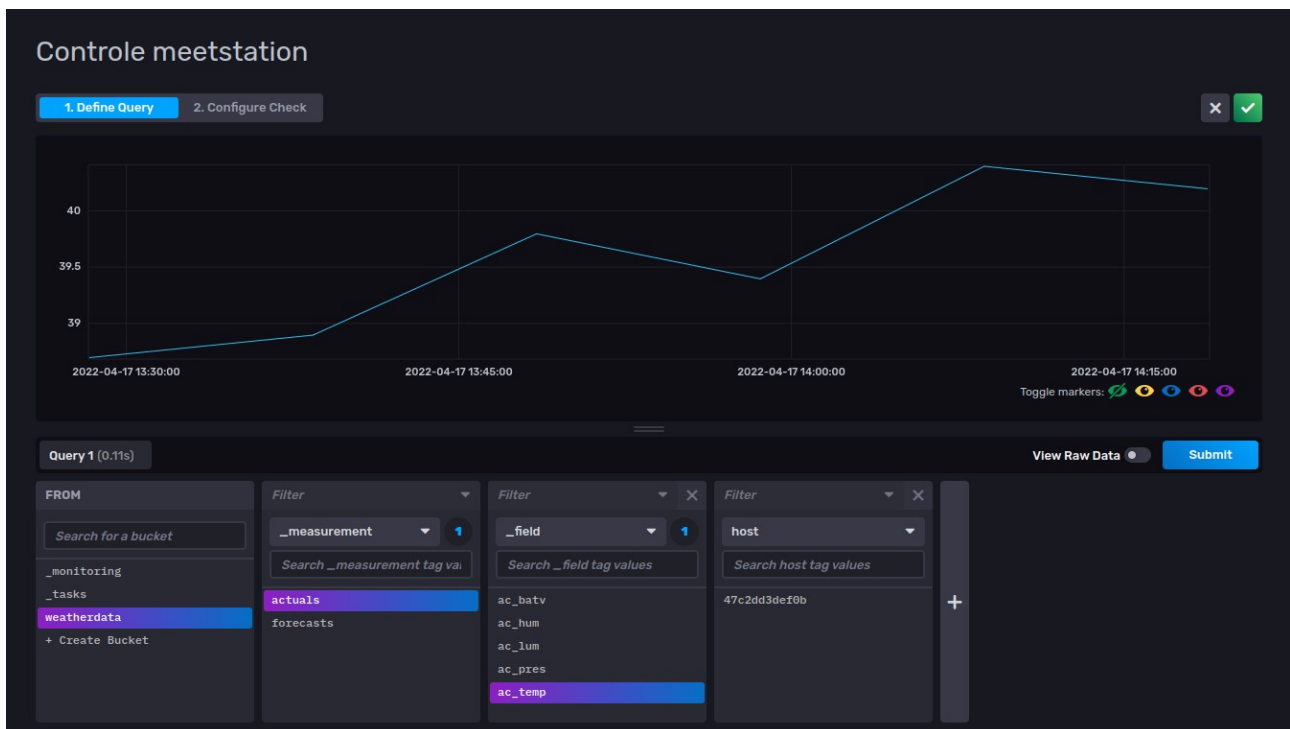
- status *OK* : spanning > 3.7
- status *WARN* : spanning tussen 3.5 en 3.7
- status *CRIT* : spanning < 3.5

We klikken op de groene check knop om de threshold check te bewaren.

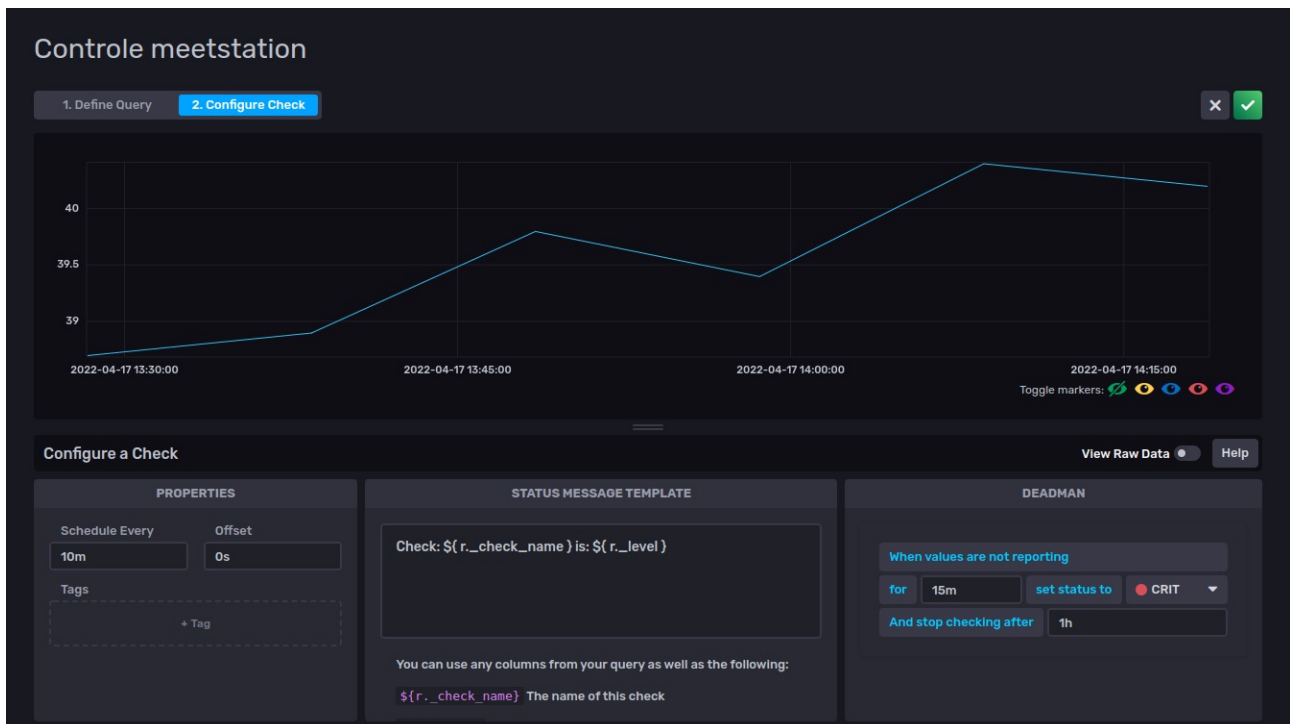
Vanaf nu wordt deze controle iedere 10 minuten uitgevoerd en de status wordt in de systeem bucket *\_monitoring* bewaard.

## Controle meetstation

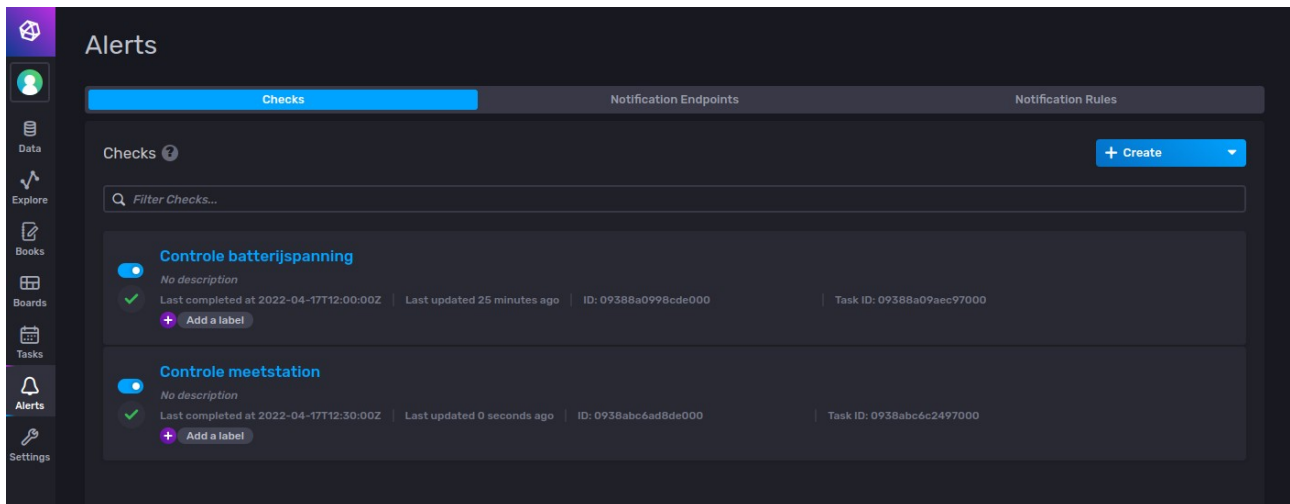
We klikken op *Alerts* en vervolgens op *Create – Deadman Check*. We selecteren **1. Define Query**. We kiezen het veld *ac\_temp* uit de tabel *actuals* van de *weatherdata* database. Klik op *Submit*



Vervolgens klikken we op **2. Configure Check**. We laten de check iedere 10m uitvoeren, dit is ook de frequentie waarmee de metingen worden gedaan. De *Deadman* configureren we als status *CRIT* wanneer 15m geen metingen binnenkomen.



We klikken op de groene check knop om de deadman check te bewaren.



Vanaf nu wordt deze controle iedere 10 minuten uitgevoerd en de status wordt in de systeem bucket `_monitoring` bewaard.

## SendGrid e-mail service

InfluxDB heeft jammer genoeg geen eigen ingebouwde email mogelijkheid, maar kan gebruik maken van diverse externe email services (SendGrid, Amazon SES, Mailjet, Mailgun). De documentatie erover vind je [hier](#).

Hier volgt verkort de set-up van de [SendGrid](#) e-mail service die een gratis service aanbiedt beperkt tot 100 e-mails per dag. Dat is ruim voldoende voor ons gebruik, daar we alleen e-mails gaan versturen bij kritische situaties.

We beginnen met een account aan te maken met een email adres en paswoord (min 16 karakters). Er worden ook een aantal persoonlijke gegevens gevraagd zoals naam, adres, firma, ... Maar geen kredietkaart nummer ;-)

Vervolgens wordt een email verstuurd om je account te verifiëren/activeren. Je kan nu inloggen op <http://app.sendgrid.com>

Voor de veiligheid wordt je de eerste keer verplicht om *dubbele authenticatie* te activeren via een app of sms.



Om de integratie met InfluxDB mogelijk te maken moet je nu een SendGrid API key genereren via *Settings – API Keys*

Geef de API Key een naam, kies de gewenste rechten (*Full Access*) en klik op de *Create & View* knop.

De API key wordt getoond. Kopieer die voorlopig op een veilige plaats. We gaan die later nodig hebben in ons InfluxDB script om e-mails te versturen.

We zullen deze API key nu opslaan in het InfluxDB *secrets* bestand. Hiervoor selecteren we in de UI interface achtereenvolgens *Settings, Secrets* en de knop *Add Secret*. Geef de Key een naam en kopieer de eerder aangemaakte SendGrid API key in het *Value* veld. Klik vervolgens op de knop *Add Secret*. Op die manier wordt geen gevoelige informatie in de e-mail scripts gebruikt, enkel de verwijzing naar de plaats waar de secrets zich bevinden.

We zijn nu klaar om onze e-mail scripts aan te maken. We zullen hiervoor *Tasks* gebruiken. `DOCKER_INFLUXDB_INIT_PASSWORD=Pwd4Fr@nk`

## Create API Key

API Key Name \*  
SendGrid\_API\_KEY

API Key Permissions \*

- ☒ **Full Access**  
Allows the API key to access GET, PATCH, PUT, DELETE, and POST endpoints for all parts of your account, excluding billing and Email Address Validation.
- ☐ **Restricted Access**  
Customize levels of access for all parts of your account, excluding billing and Email Address Validation.
- ☐ **Billing Access**  
Allows the API key to access billing endpoints for the account. (This is especially useful for Enterprise or Partner customers looking for more advanced account management.)

[Cancel](#) [Create & View](#)



### API Key Created

Please copy this key and save it somewhere safe.  
For security reasons, we cannot show it to you again

SG.qhYWtG9gQVW4IW7UbcZ-Q.woQb\_bWXLZkOu\_s6F0V-4nSZFoWOTNtrTKL3RfcC\_6Y

Done

## Add Secret

Make sure you know your secret value! You will be able to reference the secret in queries by key but you will not be able to see the value again.

Key \*

SENDGRID\_API\_KEY

Value

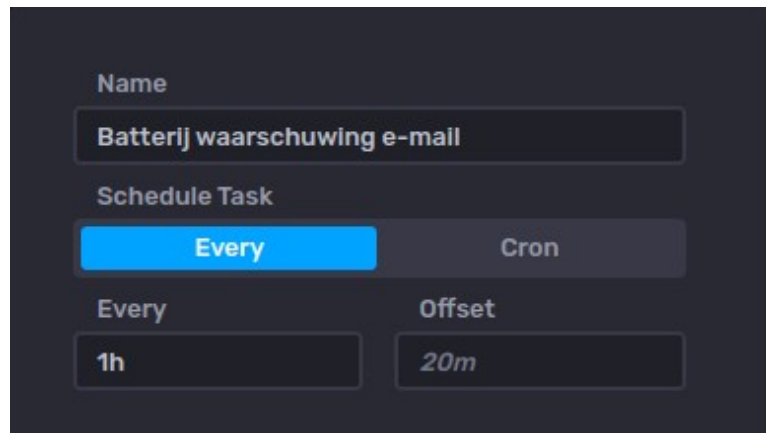
My Secret SenGrid API key :P

Cancel

Add Secret

## Batterij waarschuwing e-mail

We definiëren een nieuwe taak via *Tasks – Create Task*. We geven de taak een *Name* en een *Schedule*. In ons geval laten we de taak om het uur lopen.



In het script lezen we de SendGrid API key uit het *secrets* bestand, laten we een query lopen die het aantal kritische statussen van de batterij berekent tijdens het laatste uur en verstuurt een e-mail via de SendGrid e-mail service als dat aantal groter dan 3 is.

```
import "http"
import "json"
import "influxdata/influxdb/secrets"

option task = {name: "Batterij waarschuwing e-mail", every: 1h}

SENDGRID_APIKEY = secrets.get(key: "SENDGRID_APIKEY")

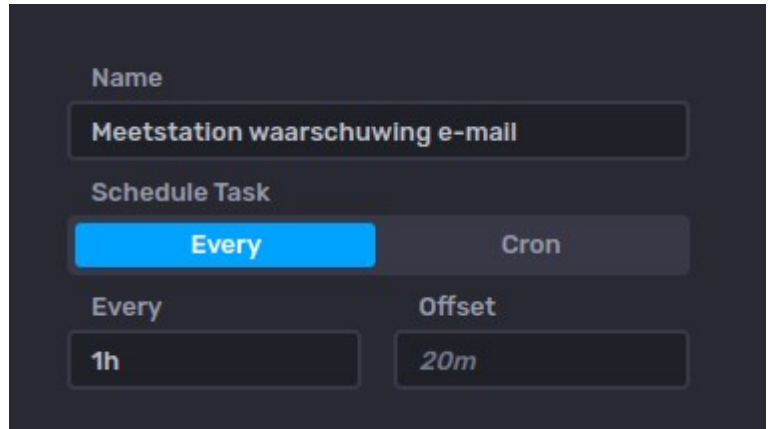
numberOfBatteryCrits =
  from(bucket: "_monitoring")
    |> range(start: -task.every)
    |> filter(fn: (r) => r._measurement == "statuses" and r._field == "ac_batv" and r._level == "crit")
    |> count()

numberOfBatteryCrits
  |> map(
    fn:
      (r) =>
        if r._value > 3 then
          {r with _value:
            http.post(
              url: "https://api.sendgrid.com/v3/mail/send",
              headers: {
                "Content-Type": "application/json",
                "Authorization": "Bearer ${SENDGRID_APIKEY}",
              },
              data:
                json.encode(
                  v:
                    {
                      "personalizations": [{to: [{email: "effevee@gmail.com"}]}],
                      "from": {"email": "effevee@gmail.com"},
                      "subject": "Effevee's weerstation waarschuwing",
                      "content":
                        [
                          {
                            "type": "text/plain",
                            "value":
                              "Opgelet: de batterijspanning van het meetstation
heeft meer dan ${r._value} kritische statussen.",
                            },
                        ],
                        1,
                      },
                    ),
                  ),
                },
              else
                {r with _value: 0},
            )
          )
        )
      )
    )
  )
)
```

*Batterij waarschuwing e-mail*

## Meetstation waarschuwing e-mail

We definiëren een nieuwe taak via *Tasks – Create Task*. We geven de taak een *Name* en een *Schedule*. In ons geval laten we de taak om het uur lopen.



In het script lezen we de SendGrid API key uit het *secrets* bestand, laten we een query lopen die het aantal kritische statussen (*dead == true*) berekent tijdens het laatste uur en verstuurt een e-mail via de SendGrid e-mail service als dat aantal groter dan 3 is.

```
import "http"
import "json"
import "influxdata/influxdb/secrets"

option task = {name: "Meetstation waarschuwing e-mail", every: 1h}

SENDGRID_APIKEY = secrets.get(key: "SENDGRID_APIKEY")

numberOfDeads =
  from(bucket: "_monitoring")
  |> range(start: -task.every)
  |> filter(fn: (r) => r._measurement == "statuses" and r._field == "dead" and r._value == true)
  |> count()

numberOfDeads
|> map(
  fn:
    (r) =>
      if r._value > 3 then
        {r with _value:
          http.post(
            url: "https://api.sendgrid.com/v3/mail/send",
            headers: {
              "Content-Type": "application/json",
              "Authorization": "Bearer ${SENDGRID_APIKEY}",
            },
            data:
              json.encode(
                v:
                  {
                    "personalizations": [{ "to": [{ "email": "effevee@gmail.com" }] }],
                    "from": { "email": "effevee@gmail.com" },
                    "subject": "Effevee's weerstation waarschuwing",
                    "content":
                      [
                        {
                          "type": "text/plain",
                          "value":
                            "Opgelet: het meetstation heeft reeds meer dan $
{r._value} maal geen gegevens verstuurd.",
                        }
                      ],
                    },
                  ),
                },
              {r with _value: 0},
            )
          }
        else
          {r with _value: 0},
        )
      )
    )
  )
```

*Meetstation waarschuwing e-mail*

Onze beide taken zijn vanaf nu actief en zullen een email versturen als de batterijspanning van het meetstation te laag wordt af als het meetstation geen gegevens meer verstuurt.

Tasks

Filter tasks... Sort by Name (A → Z) Show Inactive + Create Task

**Batterij waarschuwing e-mail**

Active Last completed at 2022-04-18T05:00:00Z Scheduled to run every 1h ID: 0938c31677c97000

+ Add a label

**Meetstation waarschuwing e-mail**

Active Last completed at 2022-04-18T05:00:00Z Scheduled to run every 1h ID: 0939a665ce097000

+ Add a label

## Effevee's weerstation waarschuwing Inbox x



effegee@gmail.com via sendgrid.net  
aan mij

zo 17 apr. 13:12 (18 uur geleden) ☆ ↩ ⋮

Opgelet: de batterijspanning van het meetstation heeft meer dan 6 kritische statussen.

↩ Beantwoorden

➡ Doorsturen

## Effevee's weerstation waarschuwing Inbox x



effegee@gmail.com via sendgrid.net  
aan mij

zo 17 apr. 13:16 (18 uur geleden) ☆ ↩ ⋮

Opgelet: het meetstation heeft reeds meer dan 6 maal geen gegevens verstuurd.

↩ Beantwoorden

➡ Doorsturen

# Bijlagen

## Datasheets

- [ESP32-WROOM-32](#) microcontroller
- [AM2320](#) temperatuur- en vochtigheidssensor
- [BMP180](#) luchtdruk- en temperatuursensor
- [BH1750FVI](#) lichtintensiteit sensor
- [TP4056](#) lithium ion batterij laadcontroller
- [DW01A](#) lithium ion batterij bescherming
- [HT73333](#) 3,3V LDO spanningsregelaar

## Referenties

- [Power ESP32/ESP8266 with Solar Panels \(includes battery level monitoring\)](#)
- [Cheap and simple Solar Power for our small Projects \(ESP32, ESP8266, Arduino\)](#)
- [ESP32 / BC24 Extending your WiFi Range](#)
- [Docker Tutorial for Beginners \[FULL COURSE in 3 Hours\]](#)
- [InfluxDB reference](#)
- [Running InfluxDB 2.0 and Telegraf Using Docker](#)
- [Handling IOT data with MQTT, Telegraf, InfluxDB and Grafana](#)
- [How to Use docker-compose Environment Variables](#)