

Voor deze opdracht moesten we kubernetes cluster running te krijgen in een windows omgeving. Dit moest dan door behulp van MQTT sensor data van een esp32 geprogrammeerd door epsHOME kunnen worden gegeven in Influxdb.

Github: <https://github.com/BryanAertgeerts/IoT-installatie.git>

Kubernetes

Eerst heb ik geprobeerd kubernetes werkend te krijgen in windows powershell. Dit kreeg ik echter niet werkend. Hierom heb het in docker gedaan. Hier kreeg ik de opdracht werkend. Om kubernetes werkend te krijgen heb ik de volgende stappen uitgevoerd:

- Vanuit het Docker Dashboard, selecteer de Instellingen.
- Selecteer Kubernetes aan de linkerzijde.
- Naast "Enable Kubernetes", selecteer het selectievakje.
- Selecteer "Apply & Restart" om de instellingen op te slaan en klik vervolgens op "Install" om te bevestigen. Dit instantieert afbeeldingen die nodig zijn om de Kubernetes-server als containers uit te voeren en installeert het /usr/local/bin/kubectl-commando op uw computer.

Hierna heb ik gezien of dit werkte door volgende commando's te doen te doen in command prompt en dit gaf aan dat het inderdaad running was.

- kubectl config get-contexts

```
C:\Users\Gebruiker>kubectl config get-contexts
CURRENT   NAME             CLUSTER           AUTHINFO           NAMESPACE
*         docker-desktop   docker-desktop    docker-desktop
```

- kubectl config get-contexts

```
C:\Users\Gebruiker>kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
docker-desktop      Ready    control-plane   6d4h   v1.25.4
```

Influxdb

Na dat kubernetes werkten heb ik influxdb in de kubernetes cluster toegevoegd. Eerst heb ik hiervoor minikube gestart in command prompt. Hierna heb ik de voorbeeld InfluxDB-configuratie toegevoegd.

- `kubectl apply -f https://raw.githubusercontent.com/influxdata/docs-v2/master/static/downloads/influxdb-k8-minikube.yaml`

Dit creëert een 'influxdb'-namespace, service en StatefulSet. Er wordt ook een PersistentVolumeClaim gemaakt om gegevens op te slaan die naar InfluxDB worden geschreven.

StatefulSet: Een StatefulSet is een object in Kubernetes waarmee je stateful applicaties kunt implementeren. Het biedt consistente toestandsinformatie door ervoor te zorgen dat elke instantie van de applicatie een unieke identiteit heeft en dat de toestandsinformatie wordt opgeslagen op een persistente schijf die is gekoppeld aan die identiteit.

PersistentVolumeClaim: Een PersistentVolumeClaim (PVC) is een Kubernetes-object dat een verzoek van een pod naar een persistent volume (PV) beschrijft. Het stelt je in staat om persistent storage aan te vragen zonder het specifieke opslagmedium te specificeren en maakt het mogelijk om gegevens op te slaan die persistent moeten zijn tussen meerdere pods of containers.

Hierna gaan we checken of de pod is running:

- `kubectl get pods -n influxdb`

```
C:\WINDOWS\system32>kubectl get pods -n influxdb
NAME          READY   STATUS    RESTARTS   AGE
influxdb-0    1/1     Running   4 (157m ago)  6d4h
```

Dan bekijken we of de service beschikbaar is door te zien naar het IP address achter Endpoints:

- `kubectl describe service -n influxdb influxdb`

```
C:\WINDOWS\system32>kubectl describe service -n influxdb influxdb
Name:          influxdb
Namespace:     influxdb
Labels:        <none>
Annotations:   <none>
Selector:      app=influxdb
Type:          ClusterIP
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.96.168.172
IPs:           10.96.168.172
Port:          influxdb 8086/TCP
TargetPort:    8086/TCP
Endpoints:     10.1.0.35:8086
Session Affinity: None
Events:        <none>
```

Als laste sturen we de port 8086 door van binnen het cluster naar localhost:

- `kubect1 port-forward -n influxdb service/influxdb 8086:8086`

```
C:\WINDOWS\system32>kubect1 port-forward -n influxdb service/influxdb 8086:8086
Forwarding from [::1]:8086 -> 8086
```

Influxdb setup

Eerst starten we influxdb op door [localhost:8086](#)

Stel uw gebruiker in:

- Voer een gebruikersnaam in voor uw eerste gebruiker.
- Voer een wachtwoord in en bevestig het wachtwoord voor uw gebruiker.
- Voer de naam van uw initiële organisatie in.
- Voer de naam in voor uw initiële bucket.
- Klik op "Doorgaan".

MQTT Broker To Kubernetes Cluster

Voor MQTT te actief te krijgen moeten we 2yaml files toevoegen.

Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: broker-depl-name
  labels:
    app: myappbroker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myappbroker
  template:
    metadata:
      labels:
        app: myappbroker
    spec:
      containers:
        - name: broker-container
          image: hivemq/hivemq-ce
          resources:
            limits:
              memory: "0.6Gi"

              cpu: "300m"
            requests:
              memory: "0.2Gi"
              cpu: "150m"
          ports:
            - containerPort: 1883
```

De bovenstaande code beschrijft een Kubernetes Deployment-object dat een container voor de HiveMQ Community Edition MQTT-broker maakt. Hieronder staan de belangrijkste onderdelen van de YAML-configuratie:

- `apiVersion`: de API-versie die wordt gebruikt voor het object, in dit geval de `apps/v1` API-versie.
- `kind`: het type object dat wordt gemaakt, in dit geval een Deployment-object.
- `metadata`: metadata voor het object, zoals de naam en labels.
- `spec`: de specificaties voor de Deployment, inclusief het aantal replica's, de selector om pods te selecteren, en de template voor het maken van pods.
- `containers`: een lijst van containers die in de pod moeten worden uitgevoerd. In dit geval wordt er één container genaamd "broker-container" gemaakt.
- `name`: de naam van de container.
- `image`: de naam van de Docker-image voor de HiveMQ-broker.
- `resources`: beperkingen en vereisten van CPU en geheugen voor de container.
- `ports`: de poort die de container beluistert voor inkomend verkeer.

De bovenstaande YAML-configuratie maakt dus een pod met één container die de HiveMQ MQTT-broker draait en de poort 1883 beluistert voor inkomend verkeer. De beperkingen en vereisten voor CPU en geheugen worden ook aangegeven.

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service-broker
spec:
  selector:
    app: myappbroker
  type: LoadBalancer
  ports:
    - port: 1883
      targetPort: 1883
      nodePort: 30005
```

De bovenstaande code beschrijft een Kubernetes Service-object dat de toegang tot de MQTT-broker mogelijk maakt via de LoadBalancer-service. Hieronder staan de belangrijkste onderdelen van de YAML-configuratie:

- `apiVersion`: de API-versie die wordt gebruikt voor het object, in dit geval de v1 API-versie.
- `kind`: het type object dat wordt gemaakt, in dit geval een Service-object.
- `metadata`: metadata voor het object, zoals de naam.
- `spec`: de specificaties voor de Service, inclusief de selector om pods te selecteren, het type van de Service, en de poorten die moeten worden beluisterd.
- `selector`: het label waarmee de pods worden geselecteerd waarnaar de Service verkeer moet doorsturen.
- `type`: het type Service, in dit geval een LoadBalancer. Dit zorgt ervoor dat de Service extern beschikbaar is via een LoadBalancer.
- `ports`: een lijst van poorten die moeten worden beluisterd en de doel-poorten waarnaar verkeer moet worden doorgestuurd.
- `port`: de poort die de Service beluistert voor inkomend verkeer.
- `targetPort`: de poort waarnaar verkeer wordt doorgestuurd voor de geselecteerde pods.
- `nodePort`: de poort op de nodes die toegang geeft tot de Service.

In dit geval maakt de YAML-configuratie een Service-object met de naam "my-service-broker", dat de MQTT-broker pods selecteert op basis van het label "app: myappbroker". De Service maakt gebruik van een LoadBalancer, zodat de Service extern beschikbaar is. Verkeer naar poort 1883 van de Service wordt doorgestuurd naar poort 1883 op de pods, en er wordt een nodePort toegewezen aan 30005.

Na dat deze yaml files waren aangemaakt moest je deze activeren. Dit heb ik gedaan door naar de file location van deze bestanden te gaan in command prompt. Hierna door de volgende comandos uit te voeren.

- `kubectl apply Deployment.yaml`
- `kubectl apply Service.yaml`

hierna kan je testen om te zien of ze online zijn gekomen.

- `kubectl get service`

```
C:\WINDOWS\system32>kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	6d5h
my-service-broker	LoadBalancer	10.101.144.228	localhost	1883:30005/TCP	5d22h

espHOME instalation

ik run espHOME op mijn raspberry pi 4. Om espHOME te installeren op raspberry pi run je de volgende comandos

- `sudo pip3 install cryptography==2.8`
- `sudo pip3 install esphome`

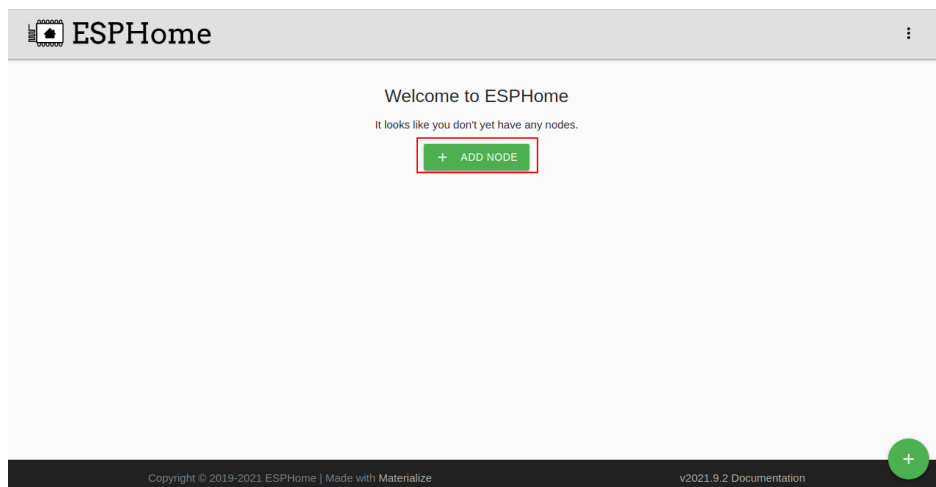
om de espHOME dashboard te starten

- `esphome dashboard config/`

espHOME devices configuration

om naar het dashboard van espHOME te gaan volg je de link <http://X.X.X.X:6052> (Vervang X.X.X.X door het IP-adres van je Raspberry Pi.)

hierna verbindt je de esp met de raspberry en klik je de “+add node” knop



Dan kan je de device een naam geven

Create configuration

Name*

This device will be configured to connect to the Wi-Fi network stored in your secrets.

CANCEL NEXT

Nu kan je je esp versie selecteren

Select your device type

Select the type of device that this configuration will be installed on.

☒ ESP32

☐ ESP32-S2

☐ ESP32-C3


☐ ESP8266

☐ Pick specific board

Pick a custom board if the default targets don't work or if you want to use the pin numbers printed on the device in your configuration.

CANCEL NEXT

Nu is het device ge configureert nu moet je hem installen dit kan wireless of wired hierna kan je hem gaan programmeren.



Configuration created!

You can now install the configuration to your device. The first time this requires a cable.

Once the device is installed and connected to your network, you will be able to manage it wirelessly.

SKIP INSTALL

How do you want to install this on your device?

Wirelessly >
Requires the device to be online

Plug into this computer >
For devices connected via USB to this computer

Plug into the computer running ESPHome Dashboard >
For devices connected via USB to the server

Manual download >
Install it yourself using ESPHome Web or other tools

CANCEL

Als je op edit klikt kan je beginnen met programmeren als je het opent staat er al basis info in.

```
1 esphome:
2   name: tes
3
4 esp32:
5   board: esp32dev
6   framework:
7     type: arduino
8
9 # Enable logging
10 logger:
11
12 # Enable Home Assistant API
13 api:
14   encryption:
15     key: "tjHidxcZv3AbsbNkWI2Ighv5vttH1c+gDuf3AeTeQ54="
16
17 ota:
18   password: "0d296faecf331f80421f37afda821675"
19
20 wifi:
21   ssid: !secret wifi_ssid
22   password: !secret wifi_password
23
24 # Enable fallback hotspot (captive portal) in case wifi connection fails
25 ap:
26   ssid: "Tes Fallback Hotspot"
27   password: "pvXyVeMvj1lM"
28
29 captive_portal:
30
```

Het definieert de naam van het apparaat ("tes"), de gebruikte ESP32 ontwikkelbord ("esp32dev") en het Arduino framework dat voor de programmatie wordt gebruikt. Het schakelt de logging in, activeert de Home Assistant API en stelt een wachtwoord in voor de OTA (Over The Air) firmware-updatefunctionaliteit. De WiFi-configuratie wordt ook ingesteld dit wordt automatisch gedaan, met behulp van geheime waarden voor het WiFi-SSID en -wachtwoord. Als de verbinding met het WiFi-netwerk mislukt, wordt een fallback-hotspot geactiveerd met de naam "Tes Fallback Hotspot" en een wachtwoord "pvXyVeMvj1lM". Tot slot wordt de captive_portal functie geactiveerd, wat betekent dat als de verbinding met het WiFi-netwerk nog steeds mislukt, de microcontroller een hotspot met een captive portal zal openen waar gebruikers zich kunnen aanmelden en het apparaat kunnen configureren.

de uiteindelijke yaml die ik heb gebruikt voor in mijn esp:

```
esphome:
  name: esphome-web-9c7f94

esp32:
  board: esp32dev
  framework:
    type: arduino

# Enable logging
logger:

# Enable Home Assistant API
api:

ota:

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

# Enable fallback hotspot (captive portal) in case wifi connection fails
ap:
  ssid: "Esphome-Web-9C7F94"
  password: "YVeteyWe2y9L"

mqtt:
  broker: "tcp://127.0.0.1:1883"
  username: "telegraf"
  password: "metricsmetricsmetricsmetrics"

captive_portal:

sensor:
  - platform: dht
    pin: GPIO4
    temperature:
      name: "Temperature garage"
      id: dht11_0_t
    humidity:
      name: "Humidity garage"
      id: dht11_0_h
    update_interval: 60s

  - platform: template
    name: "Dew Point"
    lambda: |-
      return (243.5*(log(id(dht11_0_h).state/100)+((17.67*id(dht11_0_t).state)/
      (243.5+id(dht11_0_t).state)))/(17.67-log(id(dht11_0_h).state/100)-
      ((17.67*id(dht11_0_t).state)/(243.5+id(dht11_0_t).state)))));
    update_interval: 60s
    unit_of_measurement: °C
    icon: 'mdi:thermometer-alert'

interval:
  - interval: 60s
    then:
      - mqtt.publish_json:
          topic: telegraf/host01/cpu
          payload: |-
            root["T_garage"] = id(dht11_0_t).state;
            root["H_garage"] = id(dht11_0_h).state;
```

Deze YAML is een configuratiebestand voor het ESPHome-platform voor het configureren van een ESP32-apparaat dat is verbonden met WiFi en MQTT. Hieronder staat een uitleg van elke sectie in dit bestand:

- **esphome:** Dit is de hoofdsectie waarin de naam van het apparaat wordt gespecificeerd.
- **esp32:** Dit is de sectie waarin de specifieke ESP32-configuratie wordt gedefinieerd, waaronder het type bord en het framework dat wordt gebruikt (Arduino in dit geval).
- **logger:** Hiermee wordt de logging-functionaliteit ingeschakeld.
- **api:** Hiermee wordt de Home Assistant API ingeschakeld, waarmee externe apparaten kunnen communiceren met de ESP32.
- **ota:** Hiermee wordt over-the-air (OTA) firmware-updates ingeschakeld, wat betekent dat de firmware van de ESP32 op afstand kan worden bijgewerkt zonder fysieke toegang tot het apparaat.
- **wifi:** Hier worden de WiFi-gegevens gespecificeerd, waaronder het SSID en het wachtwoord. Ook wordt er een fallback-hotspot ingeschakeld voor het geval de WiFi-verbinding mislukt.
- **mqtt:** Hier worden de MQTT-gegevens gespecificeerd, waaronder de broker-URL, gebruikersnaam en wachtwoord.
- **captive_portal:** Hiermee wordt de captive portal-functionaliteit ingeschakeld, wat betekent dat als de ESP32 niet verbonden is met WiFi, de gebruiker een tijdelijk WiFi-netwerk kan instellen om verbinding te maken met de ESP32 en de WiFi-gegevens te wijzigen.
- **sensor:** Hier worden de sensoren gespecificeerd die zijn aangesloten op de ESP32, inclusief een DHT-sensor voor temperatuur en luchtvochtigheid en een template-sensor voor het berekenen van het dauwpunt op basis van de metingen van de DHT-sensor.
- **interval:** Hier wordt aangegeven hoe vaak de metingen van de sensoren moeten worden gepubliceerd via MQTT, in dit geval elke 60 seconden. Er wordt ook aangegeven welke gegevens moeten worden gepubliceerd en onder welke MQTT-topic.