Inhaltsverzeichnis

Voraussetzungen

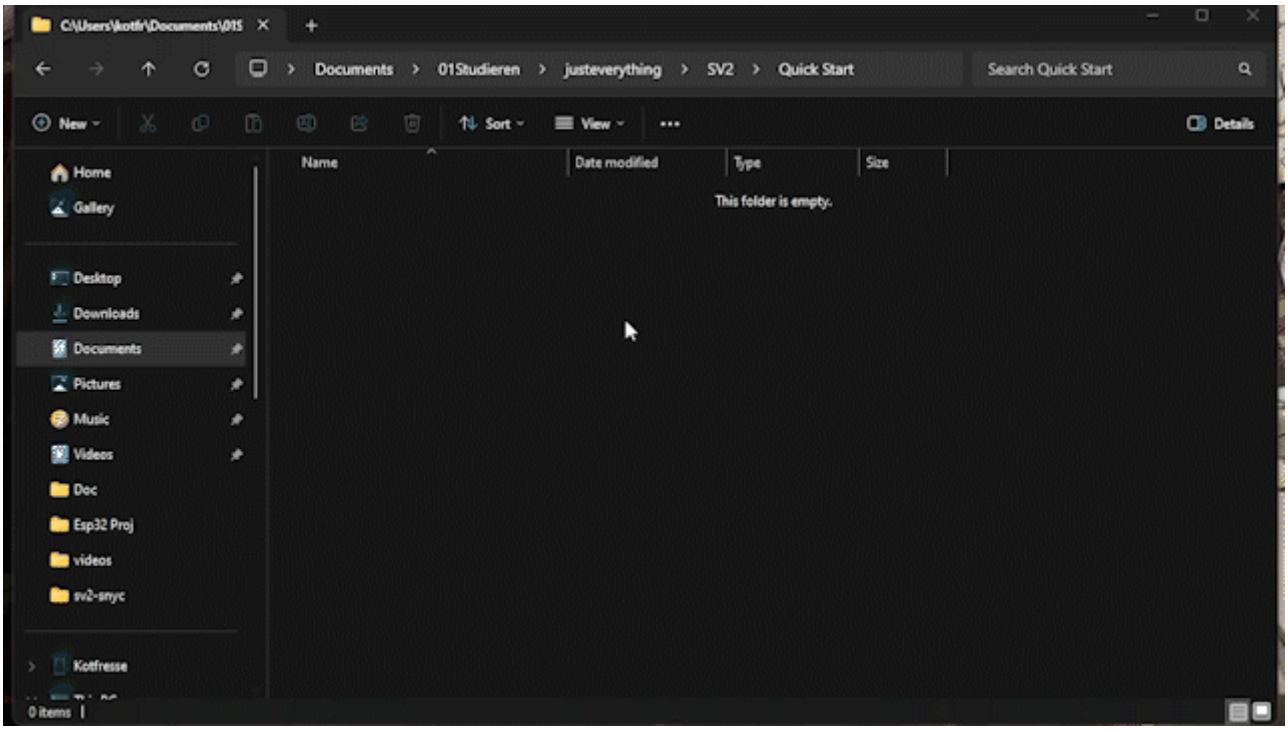
- Docker Kenntnisse

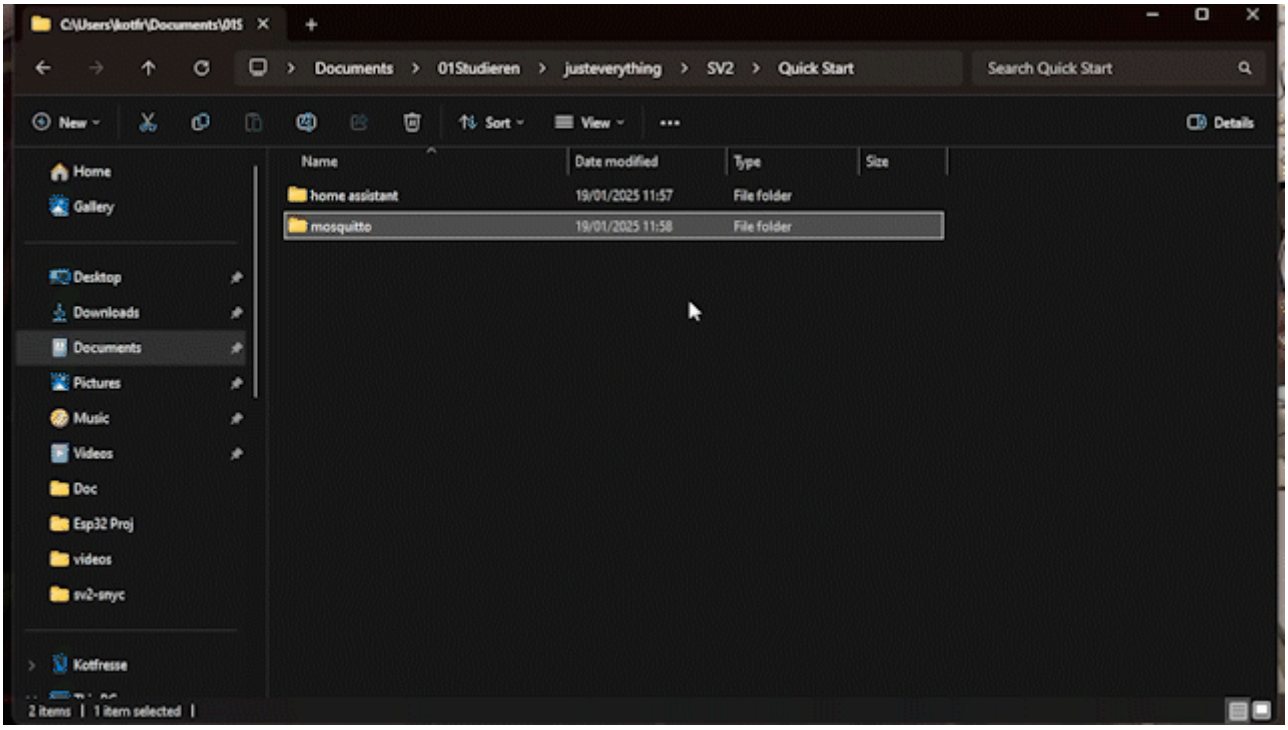
Ordnerstruktur im Verzeichnis erstellen

Erstelle eine Ordnerstruktur für das Projekt. Dies hilft, die Dateien organisiert zu halten.

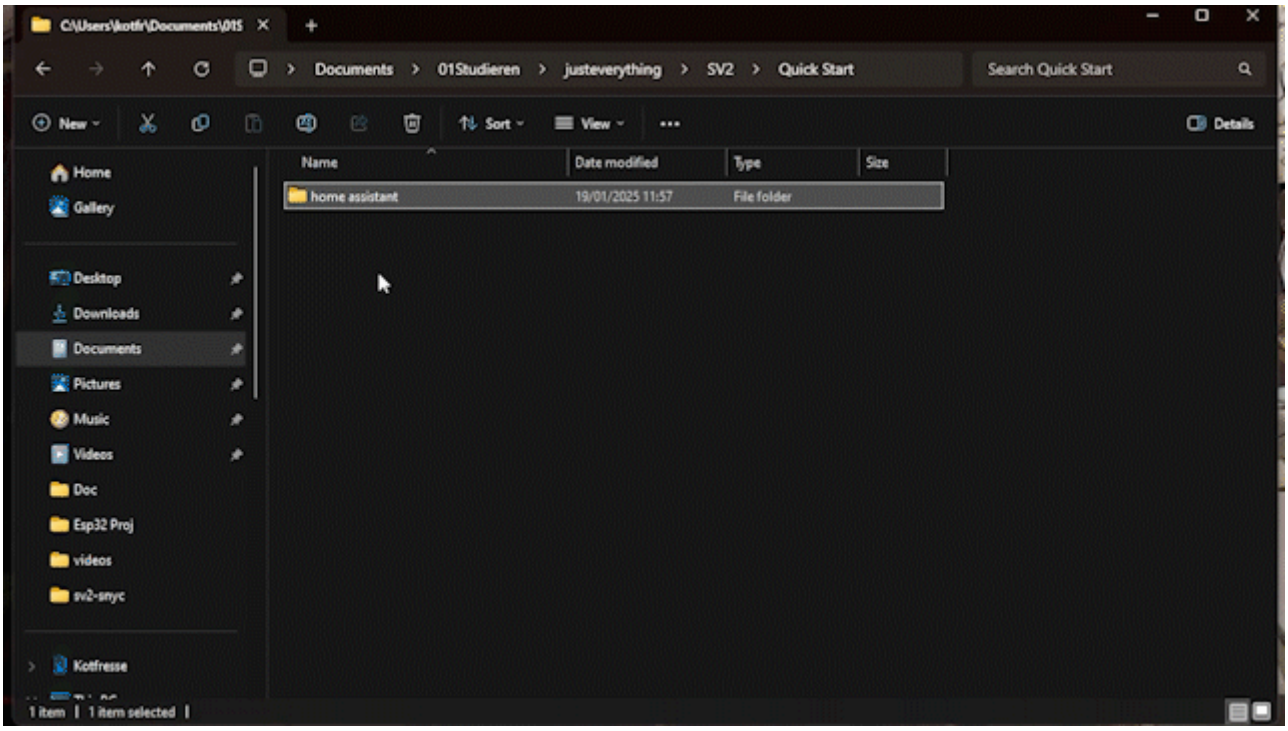
```
HHN-SV2/
├── docker-compose.yml
├── home_assistant/
│ └── config/
└── mosquitto/
  └── config/
```

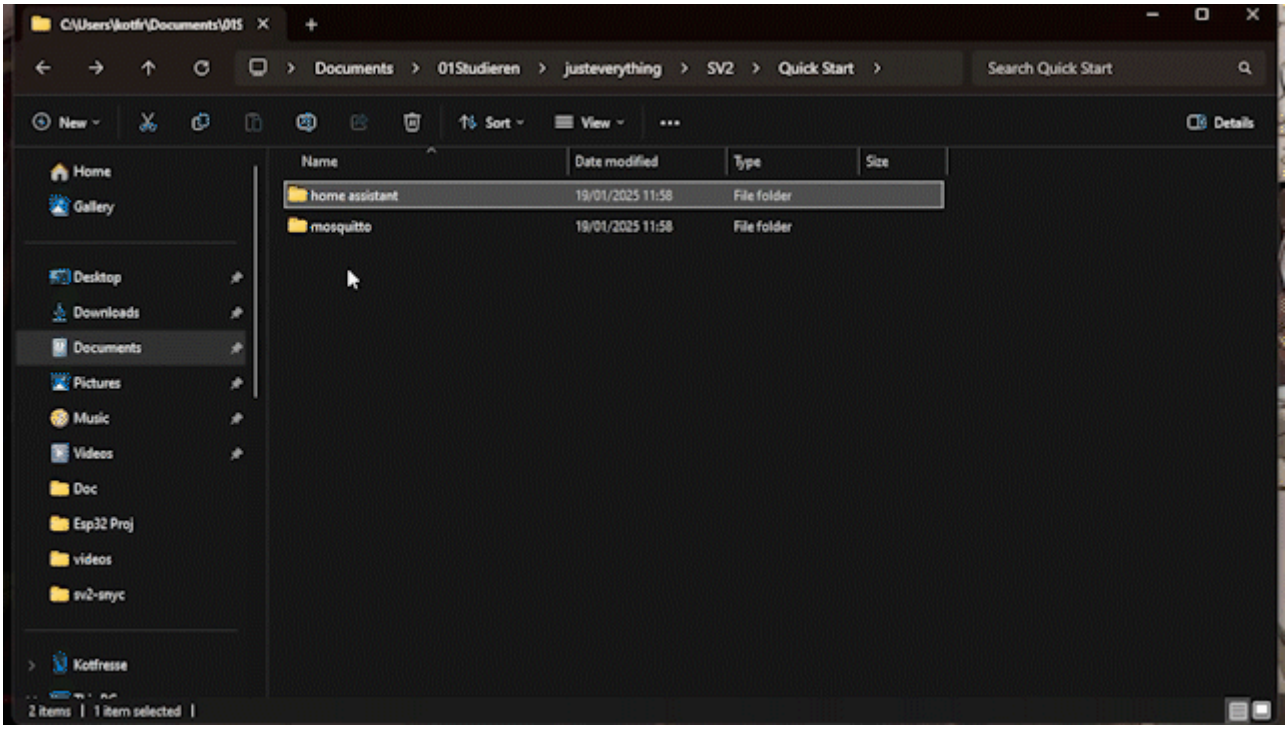
home_assistant:


C:\Users\kotfr\Documents\01S
Documents > 01Studieren > justeverything > SV2 > Quick Start
Search Quick Start
New
Sort
View
Details
Home
Gallery
Desktop
Downloads
Documents
Pictures
Music
Videos
Doc
Esp32 Proj
videos
sv2-snyc
Kotfresse
Name
Date modified
Type
Size
This folder is empty.
0 items


C:\Users\kotfr\Documents\01S
Documents > 01Studieren > justeverything > SV2 > Quick Start
Search Quick Start
New
Sort
View
Details
Home
Gallery
Desktop
Downloads
Documents
Pictures
Music
Videos
Doc
Esp32 Proj
videos
sv2-snyc
Kotfresse
Name
Date modified
Type
Size
home assistant 19/01/2025 11:57 File folder
mosquitto 19/01/2025 11:58 File folder
2 items | 1 item selected

Mosquitto:



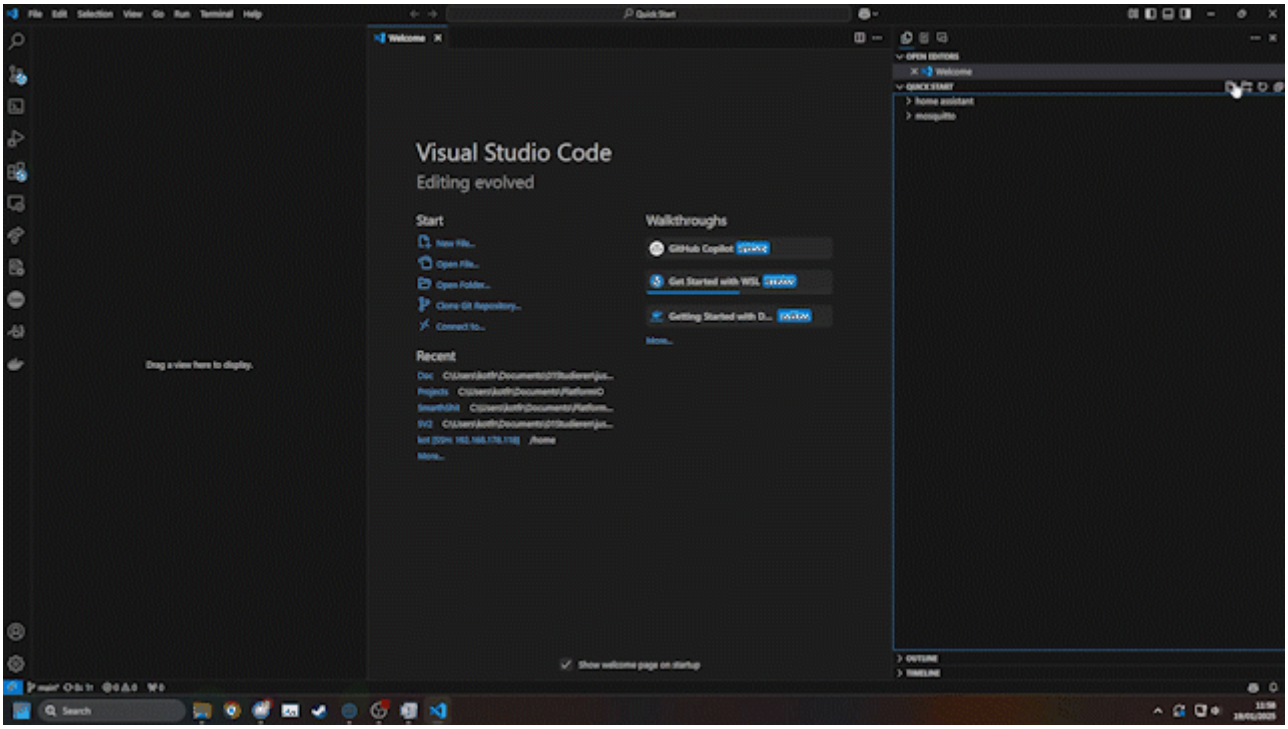


Docker-Compose Datei erstellen

Erstelle eine `docker-compose.yml`-Datei im Stammverzeichnis, um Home Assistant und den MQTT-Broker zu starten.

```
home_assistant:
  image: lscr.io/linuxserver/homeassistant:latest
  container_name: home_assistant
  restart: unless-stopped
  ports:
    - "8124:8123"
  volumes:
    - ./home_assistant/config:/config
```

```
    environment:
      - TZ=Europe/Berlin
      - MQTT_HOST=localhost
      - MQTT_PORT=1883
      - MQTT_USERNAME=koti
      - MQTT_PASSWORD=kot

  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto
    restart: always
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/config:/mosquitto/config
    environment:
      - TZ=Europe/Berlin
```



Terminal öffnen und Docker starten

1. Öffne ein Terminal im Stammverzeichnis (wo die docker-compose.yml-Datei liegt).
2. Starte die Docker-Container mit dem folgenden Befehl:

```
docker-compose up -d
```

3. Sobald alles heruntergeladen ist sollten mehr Datein in den Ordner sein.

Konfigurationsdateien ersetzen

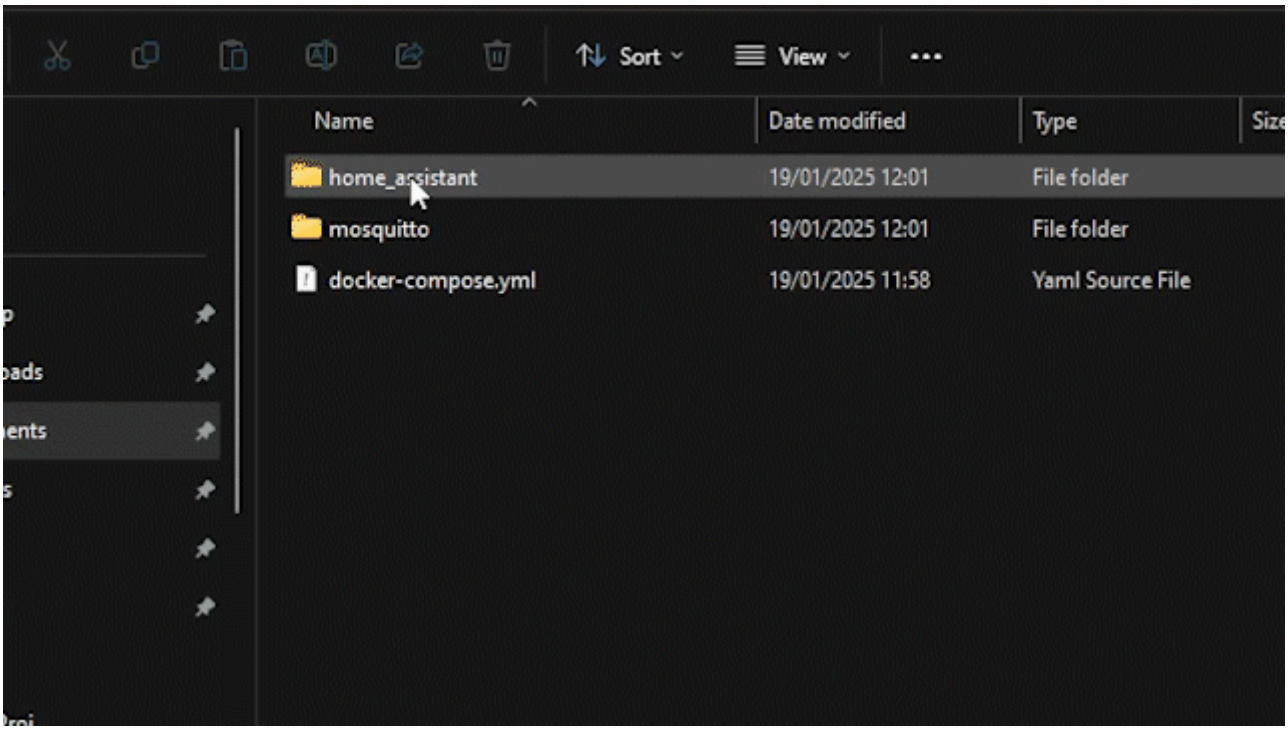
Ersetze die Konfigurationsdateien in den Ordnern home_assistant/config/ und mosquitto/config/ mit den gewünschten Einstellungen. Dies kann z. B. die Konfiguration von Home Assistant oder die MQTT-Benutzerauthentifizierung umfassen. In home_assistant/config/`configuration.yaml` haben wir:

```
# Loads default set of integrations. Do not remove.
default_config:

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

# Include automations, scripts, and scenes
automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml

# HTTP Configuration for reverse proxy or Cloudflare
http:
  use_x_forwarded_for: true
  trusted_proxies:
    - 127.0.0.1
    - 192.168.178.79 //Hier die IP von dem Gerät welches die Instanz hosted
    - ::1


# MQTT Configuration
mqtt:
  switch:
```

```
    - name: "My Led Strip"
      state_topic: "home/esp32/relay/state"
      command_topic: "home/esp32/relay/set"
      payload_on: "ON"
      payload_off: "OFF"
      state_on: "ON"
      state_off: "OFF"
      qos: 1
      retain: true

    - name: "LOFF/LON Switch"
      state_topic: "home/esp32/relay/state"  # Topic to receive the state (LOFF or
LON)
      command_topic: "home/esp32/relay/set2"  # Topic to send commands (LOFF or
LON)
      payload_on: "LON"  # Payload to send when turning on
      payload_off: "LOFF"  # Payload to send when turning off
      state_on: "LON"  # State value that represents "on"
      state_off: "LOFF"  # State value that represents "off"
      qos: 1
      retain: true

# Input Text for Color Picker
input_text:
  color_picker:
    name: Color Picker
    initial: "FFFFFF"  # Default value, ensures the field is never empty
    min: 6             # Minimum length of 6 characters
    max: 6             # Maximum length of 6 characters
    pattern: "^[0-9A-F]{6}$"  # Only allows 6 characters of 0-9 and A-F (uppercase
only)

# Input Number for Slider and Number of Turns
input_number:
  number_field:
    name: Number of Turns
    initial: 0       # Start at 0
    min: -12         # Minimum value
    max: 12          # Maximum value
    step: 1          # Step size (whole numbers only)
    mode: slider     # Display as a slider in the UI

  slider_value:
    name: Motor Speed
    initial: 5       # Default value (midpoint of 1-10)
    min: 1           # Minimum value
    max: 10          # Maximum value
    step: 1          # Step size (whole numbers only)
    mode: slider     # Display as a slider in the UI

# Input Buttons
input_button:
  send_mon_button:
    name: Send mON
```

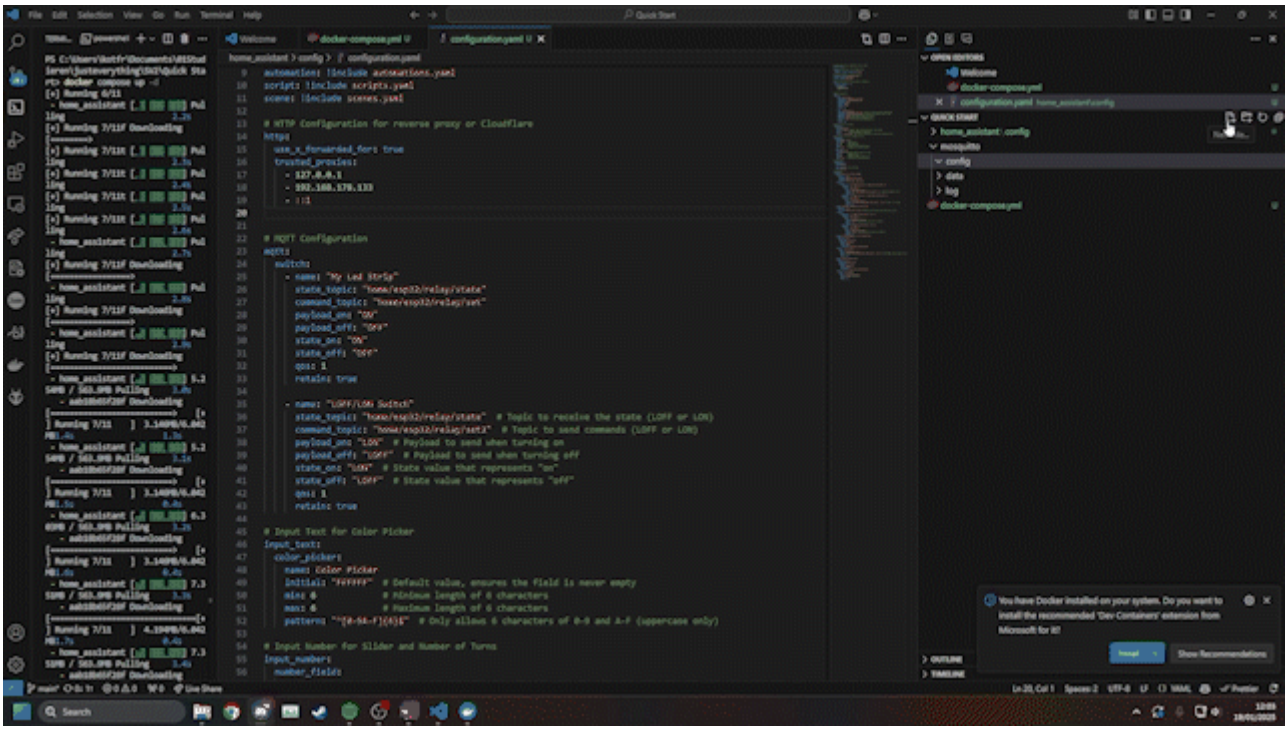
```
  send_sbn_button:  # Corrected slug
    name: Send sON

# Automations
automation:
  - alias: Send Color or Slider via MQTT
    trigger:
      - platform: state
        entity_id: input_text.color_picker
      - platform: state
        entity_id: input_number.slider_value
    action:
      - choose:
          - conditions: "{{ trigger.entity_id == 'input_text.color_picker' }}"
            sequence:
              - service: mqtt.publish
                data:
                  topic: "home/esp32/relay/set"
                  payload: "#{{ states('input_text.color_picker') }}"  # Add the
hashtag for color
                  qos: 1
                  retain: true
          - conditions: "{{ trigger.entity_id == 'input_number.slider_value' }}"
            sequence:
              - service: mqtt.publish
                data:
                  topic: "home/esp32/relay/set"
                  payload: "s{{ states('input_number.slider_value') | int }}"  #
Add 's' for slider
                  qos: 1
                  retain: true
  - alias: "Send LON or LOFF based on switch state"
    trigger:
      - platform: state
        entity_id: switch.loff_lon_switch  # Replace with the actual entity ID of
your switch
    action:
      - choose:
          - conditions: "{{ trigger.to_state.state == 'on' }}"
            sequence:
              - service: mqtt.publish
                data:
                  topic: "home/esp32/relay/set2"
                  payload: "LON"  # Send LON when the switch is turned on
                  qos: 1
                  retain: true
          - conditions: "{{ trigger.to_state.state == 'off' }}"
            sequence:
              - service: mqtt.publish
                data:
                  topic: "home/esp32/relay/set2"
                  payload: "LOFF"  # Send LOFF when the switch is turned off
                  qos: 1
```

```
                  retain: true
  - alias: Send mON and Number of Turns on Button Press
    trigger:
      - platform: event
        event_type: call_service
        event_data:
          domain: input_button
          service: press
          service_data:
            entity_id: input_button.send_mon_button
    action:
      - service: mqtt.publish
        data:
          topic: "home/esp32/relay/set"
          payload: "MON{{ states('input_number.number_field') | int }}"  # Send
'MON' and the number of turns
          qos: 1
          retain: true

  - alias: Send sON on Button Press
    trigger:
      - platform: event
        event_type: call_service
        event_data:
          domain: input_button
          service: press
          service_data:
            entity_id: input_button.send_sbn_button
    action:
      - service: mqtt.publish
        data:
          topic: "home/esp32/relay/set"
          payload: "AON"  # Send 'sON'
          qos: 1
          retain: true
```



und in mosquitto/config/`mosquitto.conf` :

```
# General settings
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log

# Default listener for MQTT
listener 1883
allow_anonymous false
password_file /mosquitto/config/passwords.txt

# WebSocket listener (optional)
listener 9001
protocol websockets

log_dest stderr
```

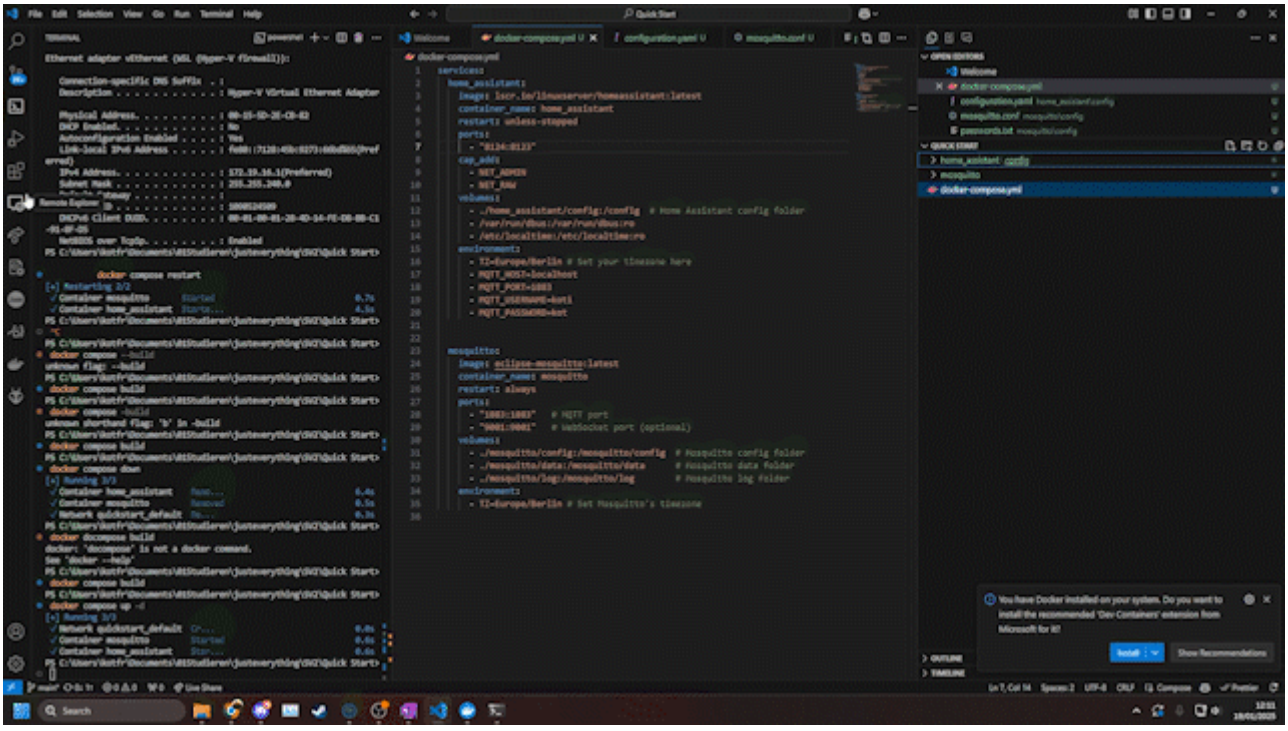


Visual Studio Code öffnen

Öffne Visual Studio Code um mit der Entwicklung des ESP32-Codes zu beginnen.

PlatformIO installieren

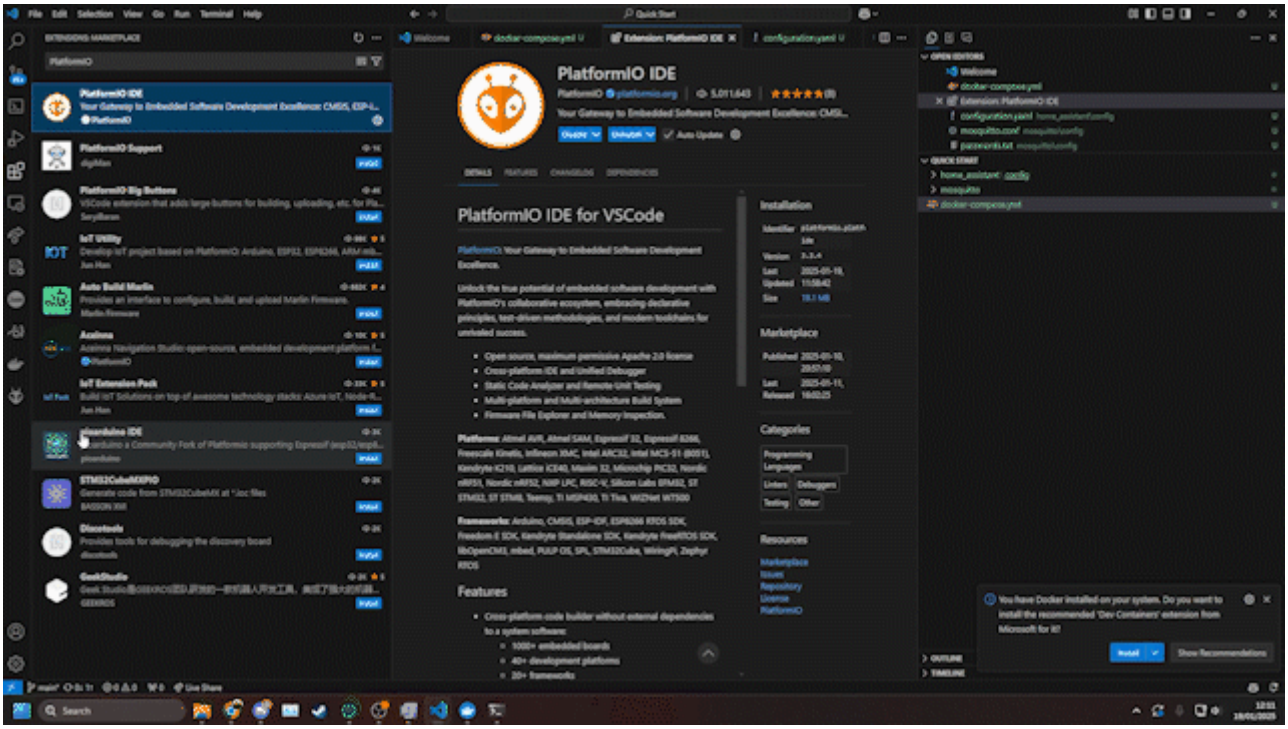
1. Öffne die Extensions-Ansicht in Vs Code.
2. Suche nach `PlatformIO` IDE und installiere es.



3. (Optional) Visual Studio Code nach der Installation neustarten.

Neues Projekt starten

1. Klicke in Vs Code auf das PlatformIO-Symbol in der Sidebar.
2. Wähle `New Project`



3. Gib dem Projekt einen Namen, z.B. ESP32_SmartHome.
4. Wähle des Board und das Framework aus.

Board auswählen

1. Falls Marc dir die Hardware gegeben hat ist es wahrscheinlich ein ESP32 Dev Module.
2. Wähle als Framework Arduino.

INI-Datei bearbeiten

Location:

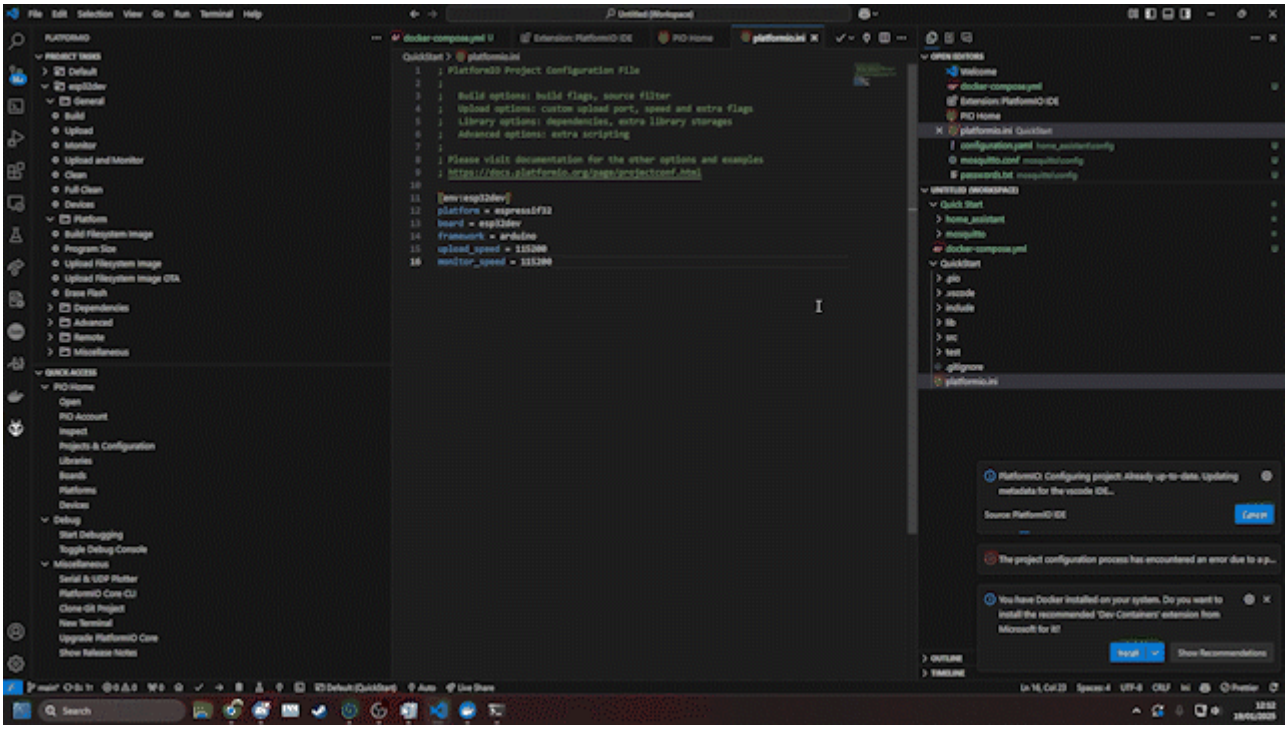
Öffne die `platformio.ini`-Datei und passe sie an:

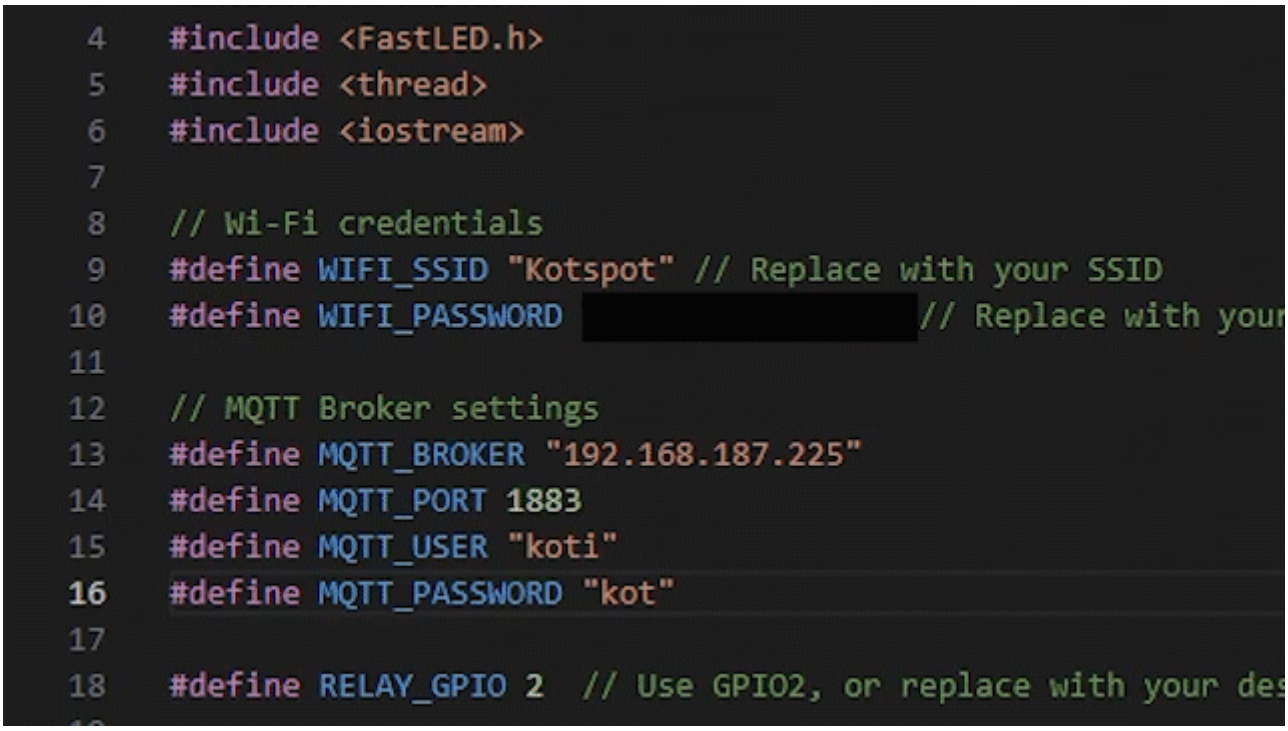
```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
upload_speed = 115200
monitor_speed = 115200
```

Code einfügen / bearbeiten

Füge den Code in die src/`main.cpp` (oder `main.c` für Espressif). z.B:

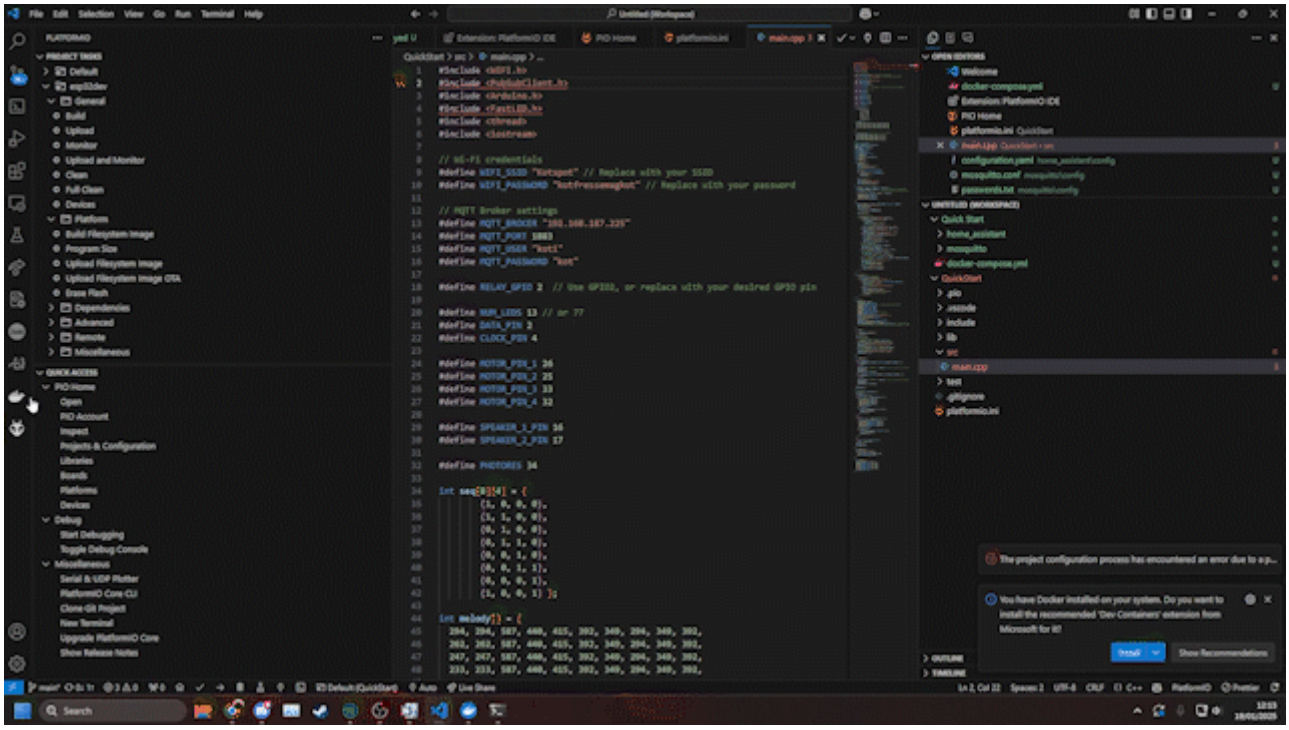
```
#include <Arduino.h> //Libraries

void setup() { //Was ausgeführt wird beim Programm start
//stuff
}

void loop() { //Wird regelmäßig wiederholt ausgeführt
//other stuff
}
```





(optional) Bibliotheken installieren

1. Öffne das PlatformIO-Symbol und klicke auf Libraries.
2. Suche nach deiner Libary und füge sie zu deinem Projekt hinzu.



3. Deine Ini file wird automatisch verändert.

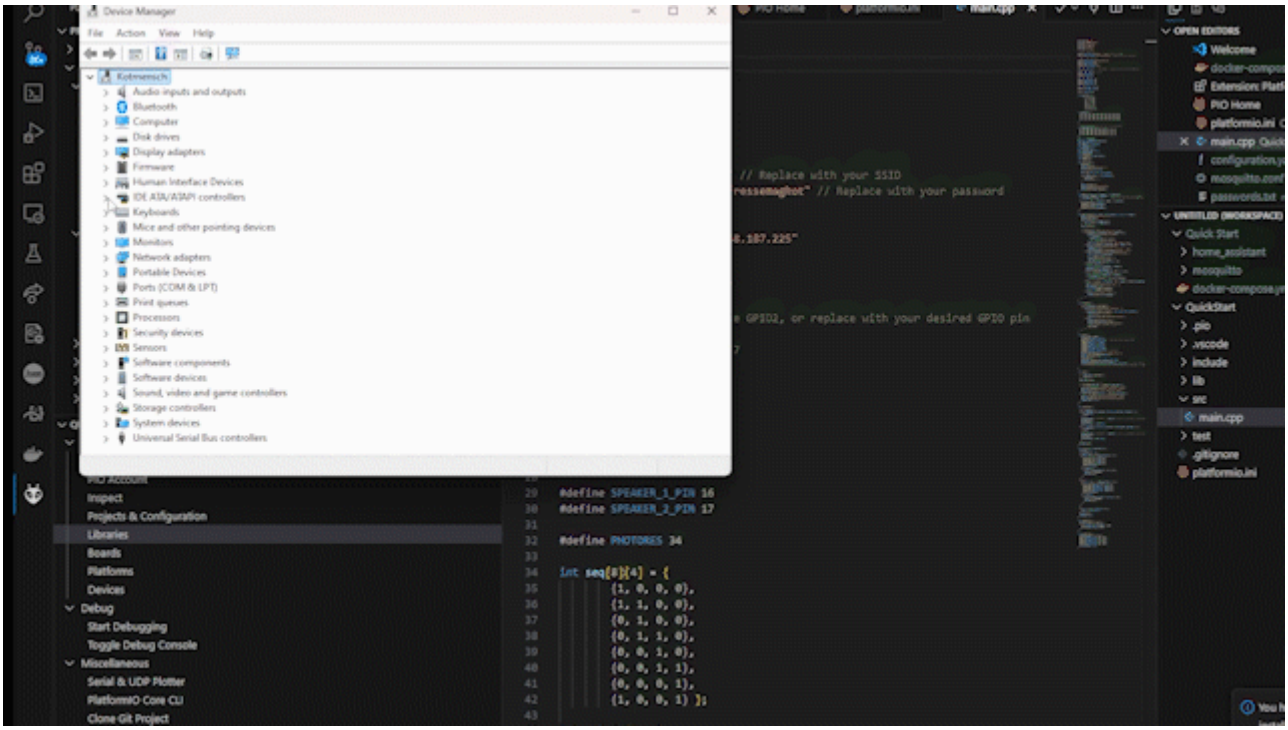
PlatformIO Terminal starten

1. Öffne das PlatformIO-Symbol und klicke auf `New Terminal`.
2. Stelle sicher, dass der ESP32 mit dem Computer verbunden ist. Du kannst es prüfen, indem du den `Geräte-Manager` öffnest und nach `Ports` schaust. Falls es nicht erkannt wird kannst du hier die Treiber hier installieren.

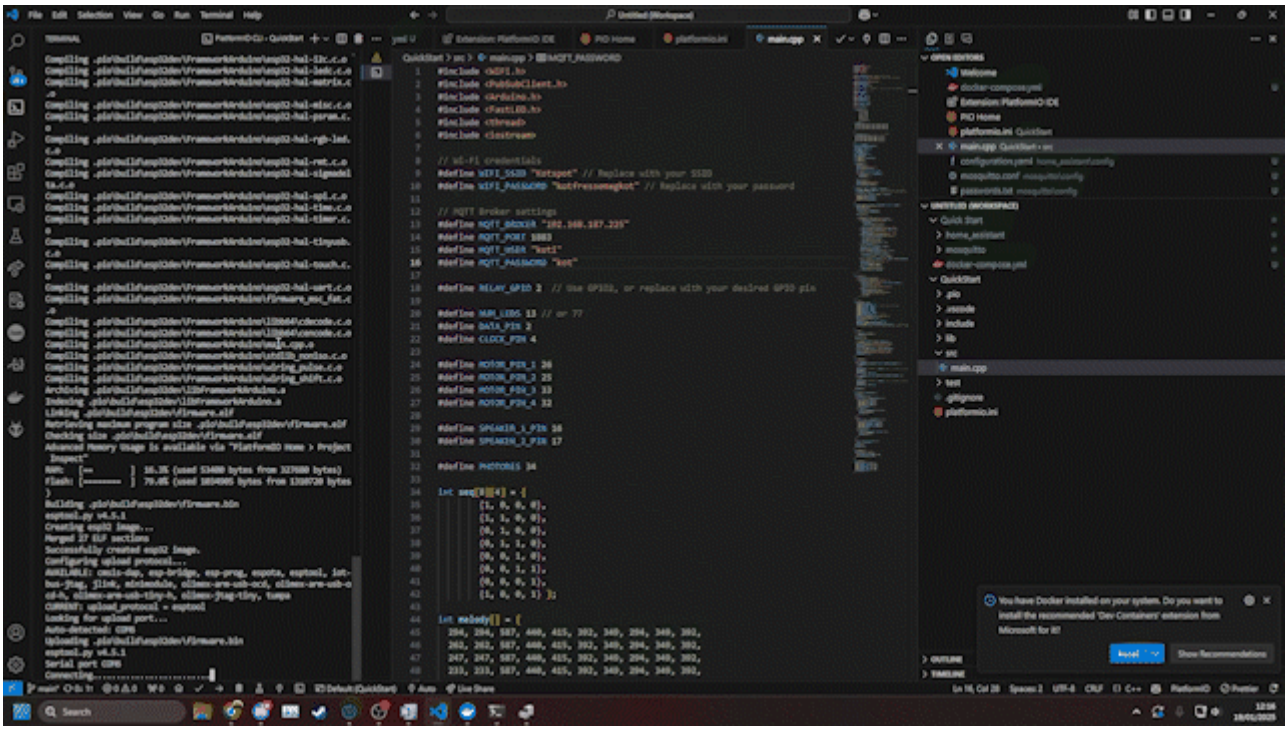
Code auf den ESP32 laden

1. Führe den folgenden Befehl im PlatformIO-Terminal aus, um den Code auf den ESP32 zu laden:

```
pio run --target upload
```



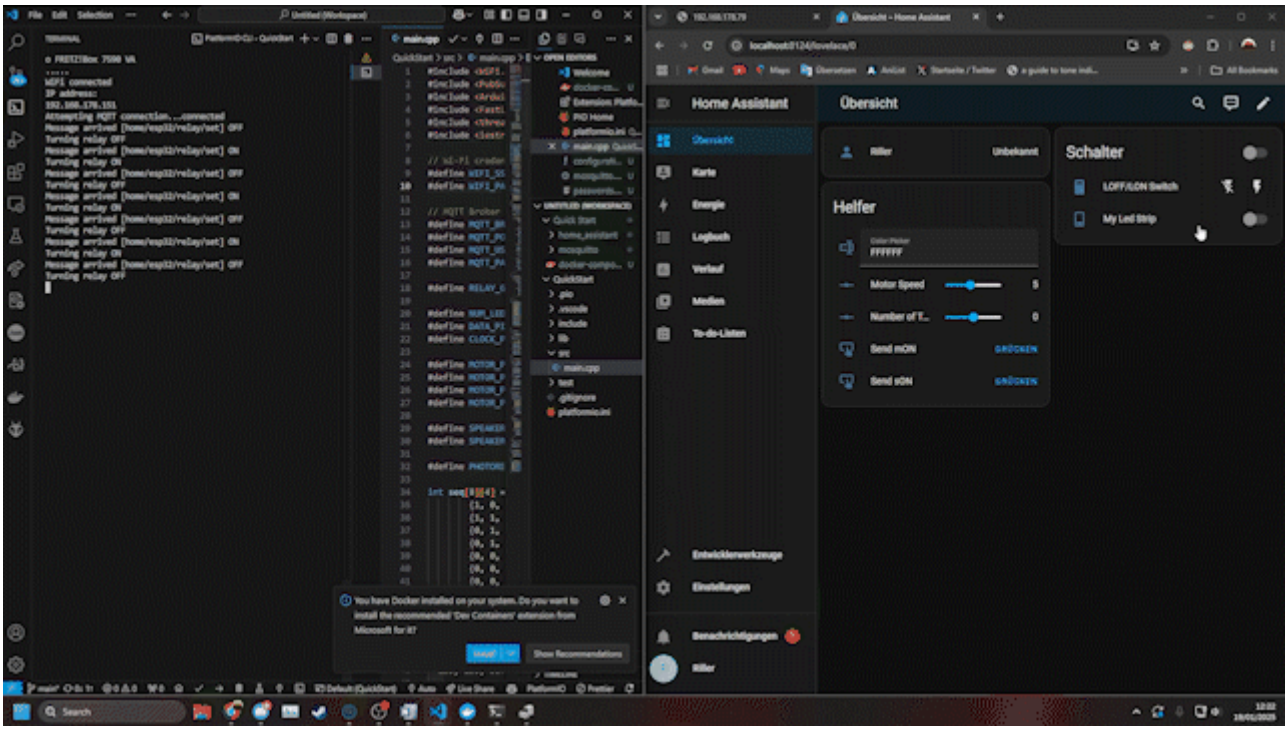
2. Es kann sein, dass der ESP32 nicht in dem richtigen Zustand ist, weshalb man während des Upload den `Boot`-Knopf, der sich auf dem Brett befindet, drücken muss.



(optional) Seriellen Monitor starten

1. Starte den seriellen Monitor, um die Ausgaben des ESP32 zu sehen:

```
pio device monitor
```



2. Der Code läuft auch ohne diesen Schritt.

Fertig

Der ESP32 führt nun den Code aus und in unserem Fall hat er sich mit dem Wlan und MQTT Broker verbunden und hat auf Nachrichten reagiert.